LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SOLO PROGRAMADORES

AÑO VI, NÚMERO 53

UNA PUBLICACIÓN DE: TOWER COMMUNICATIONS, S.R.L.

975 Ptas.

CREAR UN CHAT CON JAVA

CLIENTE Y SERVIDOR

NUEVAS TÉCNICAS DE BASES DE DATOS

Oracle, Informix, PostgreSQL

MULTIMEDIA

Reconocimiento de voz: E-mails

PROGRAMACIÓN GRÁFICA

Construir un Z buffer

HERRAMIENTAS

Novedades de Visual Basic 6.0

HTML DINÁMICO

Áreas desplegables

LINUX

Planificación de procesos

CONTENIDO DEL CD-ROM

- CLASSBUILDER 2.0
- MACROMEDIA DREAMWEAVER 2.0
- SYBASE POWERBUILDER 6.0
- SUN JDK 1.2
- NETWORKCHARTER/PRO 3.0
- SOFTQUAD HOTMETAL PRO 5.0

8 413042 303299 00053

SÓLO PROGRAMADORES

Número 53
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS

Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Subdirector
Oscar Rodríguez Fernández
oscarrf@towercom.es
Coordinador Técnico
Eduardo De Riquer Frutos
eriquer@towercom.es
Coordinadora de Redacción
Erika de la Riva Arnáiz
eriva@towercom.es

Colaboradores
Constantino Sánchez, Juan M. Menéndez, Jordi Agost, Javier Sanz, Adolfo Aladro, Enrique de la Lastra.

Maquetación y Tratamiento de Imagen
Ana Isabel Madero Bocos

Publicidad
Olga de Oteyza (Madrid)
Tel.: (91) 661 42 11
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Alicia Zazo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Laboratorio
Javier Amado (Jefe)
jamado@towercom.es
Servicio Técnico
Manuel Hernando
mhernando@towercom.es

Preimpresión
Indes Color
Impresión
Gráficas Muriel
Distribución
SGEL
Distribución en Argentina / Chile / Colombia / México / Venezuela
Capital: Huesca y Sanabria
Interior: D.G.P.

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
carmina@towercom.es
Distribución
Almiro Sanguino

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

La revista Sólo Programadores no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados. El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 30-4-99

La otra generación del 98 (de Windows 98)

Después de los cambios, novedades y ajustes llega la calma y también el momento de seguir compartiendo los acontecimientos informáticos de cada día. En este caso y coincidiendo con las fiestas navideñas, la historia tiene que ver con una madre y un pequeño que no llegaba al metro de estatura. Ante la avalancha de caprichos la indiferencia de la madre iba en aumento, sorprendida por tan onerosos inventos y por el impresionante control de su pequeño.

En un momento determinado el chico se lanzó al ataque y rogó/sugirió/exigió (no me quedó del todo claro) a su madre la compra de un nuevo ordenador, obteniendo una respuesta entre airada y alucinada: "Pero si te compramos uno nuevo hace menos de un año". El genio sonrió, conmovido por la mezcla de ignorancia e inocencia de su madre ajeno a los avances informáticos y le respondió: "Sí claro, pero con ese no puedo jugar a nada, sin Pentium II, ni tarjeta aceleradora 3D, ni *AGP*, ni *SDRAM*, ni nada de nada".

Aunque reconozco mi falta de tacto y sobre todo mi inoportunidad, no pude contener la risa ante una respuesta tan llena de razón. Una mirada inquisidora se plantó sobre mí y con el fin de aliviar la situación no se me ocurrió otra cosa que sugerir: "Lo mejor es que le compre a *Deep Blue*". Sobran comentarios sobre el grado de desinterés que recibió mi proposición, menos comprendida que si hubiera sido deshonesta.

Lo curioso es que hace no muchos años los niños nos emocionábamos delante de escaparates de juguetes y ahora resulta que con bastantes años más, somos como ellos, o al menos compartimos algunas aficiones. ¿Sobre qué hablarán durante el recreo? ¿De *drivers*, polígonos, texturas, filtrados y *buffers*? Desde luego parece que así es y me alegro de ello, aunque también espero que no sea su única inquietud y que simplemente acaben más preparados que nosotros.

Cuando descubro en las revistas los avances actuales y los comparo con los que existían cuando éramos niños me pregunto dónde se encontrará el límite, si es que lo hay. Pero sobre todo, pienso cuanto me gustaría tener una máquina del tiempo para ver esos acontecimientos, aunque sólo fuera como en la televisión, sin estar allí. De hecho espero con ansiedad y tengo la esperanza de que esos niños crezcan o que al menos encuentren tiempo de fabricarla entre partida y partida del juego de sus amores.

En fin, llamándome Edu y en estas fiestas, no estoy seguro de desearles feliz Navidad y que me tomen en serio, sin recordarles ningún anuncio del momento. En cualquier caso espero contar con todos el año que viene, lleguen o no al metro de estatura.

solo PROGRAMADORES 53

6 Noticias
NOVEDADES

Toda la información que necesitas está a tu disposición en esta interesante y actual sección de noticias que todos los meses te ofrecemos, para que no te quedes atrás en el mundo de la programación.

10
CONTENIDO DEL CD-ROM

Este mes, y para mantenernos en nuestra línea de ofrecer siempre productos que os interesan y hacen vuestra vida mucho más fáci,l os entregamos programas como NetworkCharter/Pro 3.0, PowerBuilder 6.0, Macromedia Dreamweaver 2.0, JavaScript Buttons II 1.5, Softquad HotMetal Pro 5.0 , Sun JDK 1.2 , así como las fuentes de nuestros artículos que os ayudarán a realizar las prácticas.

14 www
HTML DINÁMICO (III). AREAS DESPLEGABLES

Continuando con nuestra serie de artículos entorno al HTML Dinámico vamos a profundizar en todos los conocimientos adquiridos hasta el momento, al mismo tiempo que vamos a creamos un nuevo y útil efecto. Como de costumbre partiremos de la librería que hemos ido creando a lo largo de esta serie para sacarle todo el jugo posible.

19 Programación Multimedia
RECONOCIMIENTO DE VOZ (VI)

Tal y como os prometimos en el artículo del mes anterior, aquí tenéis la fórmula para crear un programa encargado de recoger los E-mails.

26 Java
CREACIÓN DE UN CHAT

Conversar mediante el teclado con otras persona sabiendo que pueden estar en cualquier parte de la Red es posible gracias a los programas de chat. Si a esto le añadimos la potencia del lenguaje Java para crear desarrollos de comunicaciones, la construcción de nuestro propio chat pasa a ser una tarea relativamente sencilla. Y esto es lo que vamos a explicar en nuestro capítulo de este mes.

38
Linux
PLANIFICACIÓN DE PROCESOS

Linux posee un eficaz sistema de planificación de procesos basado en tres colas, dos de ellas para procesos de tiempo real y la tercera para otros tipos de procesos. En este artículo se verá de una forma práctica cómo se puede trabajar con las prioridades, no sin antes explicar unos conceptos básicos sobre procesos.

45
Programación Gráfica
GLIDE (II)

En esta nueva entrega crearemos nuestro primer polígono en Glide mediante sombreado Gouraud para los vértices. Además, también abordaremos los pasos necesarios para construir un Z buffer.

58
Windows 95/98/NT
WINDOWS SCRIPTING HOST (II)

En el artículo anterior presentamos Windows Scripting Host ahora vamos a tratar de volcarnos especialmente en el lado práctico de WSH: cómo crear un directorio, cómo acceder a una unidad, cómo copiar un fichero, cómo leer un archivo de texto. En la medida de nuestras posibilidades intentaremos cubrir el número máximo de ejemplos.

64
Herramientas de Programación
VISUAL BASIC 6.0

Os presentamos todas las novedades que incorpora el recién aparecido Visual Basic 6.0. La definición de lo que es el producto podría englobarse en una herramienta de desarrollo rápido de aplicaciones.

72
Bases de Datos
NUEVAS TÉCNICAS DE BASES DE DATOS

En este interesante a la vez que práctico artículo se van a mostrar muchas de las nuevas técnicas con las que crear bases de datos. Veremos que empresas como Informix, Postgres, Oracle o Microsoft plantean soluciones a este tema.

52
Programación Internet CREACIÓN DE UN BUSCADOR WEB (II)

En este nuevo capítulo de Programación Internet se conocerá el corazón de un Buscador Web que no es ni más ni menos que un programa denominado Robot. Éste se encarga de analizar todos los posibles enlaces de una página HTML, obtener algún tipo de información sobre las páginas y archivos en ellas referidos, y continuar analizando las páginas enlazadas.

78
Libros
ACTUALIDAD

Los libros que os presentamos este mes estamos seguros que serán de vuestro agrado. Os ofrecemos libros con temas como por ejemplo la programación orientada a objetos, un manual sobre 3D Studio Max 2.5, el increíble manual de la última versión de Visual Basic y por último cómo desarrollar bases de datos en Java. Si te atreves con cualquiera de estos temas no dudes en consultar esta sección.

Un robusto servidor para la difusión y creación de informes integrados

BUSINESS OBJECTS ANUNCIA BROADCAST SERVER

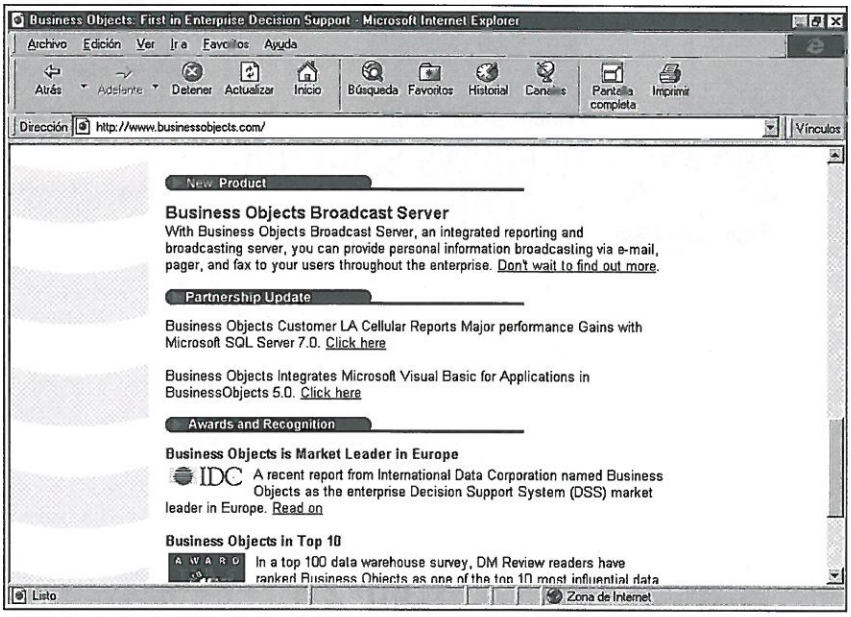
La empresa Business Objects anuncia el lanzamiento de Business Objects BROADCAST SERVER, un robusto-servidor pensado para la difusión y creación de informes integrados.

Con él los clientes pueden distribuir información personal a través de E-mail, buscapersonas o fax, a todos sus usuarios en el marco corporativo en el que se encuentren.

BROADCAST SERVER es un robusto servidor multinivel, que permite a los usuarios no técnicos editar y difundir de manera fácil y rápida, informes de datos corporativos previos o redactados expresamente. Además de esto también permite a los usuarios beneficiarse de tres ventajas claves como son la escalabilidad empresarial, la difusión de la información personal y un coste total de mantenimiento muy bajo.

Gracias a la escalabilidad se puede atender a todas las necesidades de los usuarios de una empresa, tiene una robusta arquitectura construida con Object Request Broker de CORBA.

BROADCAST SERVER le va a permitir generar y difundir todos los informes que hayan sido planificados regularmente. Asimismo también puede distribuiir automáticamente


Business Objects: First in Enterprise Decision Support - Microsoft Internet Explorer
Archivo Edición Ver Ir a Favoritos Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Historial Canales Pantalla completa Imprimir
Dirección http://www.businessobjects.com/
Vínculos
New Product
Business Objects Broadcast Server
With Business Objects Broadcast Server, an integrated reporting and broadcasting server, you can provide personal information broadcasting via e-mail, pager, and fax to your users throughout the enterprise. Don't wait to find out more.
Partnership Update
Business Objects Customer LA Cellular Reports Major performance Gains with Microsoft SQL Server 7.0. Click here
Business Objects Integrates Microsoft Visual Basic for Applications in BusinessObjects 5.0. Click here
Awards and Recognition
Business Objects is Market Leader in Europe
IDC A recent report from International Data Corporation named Business Objects as the enterprise Decision Support System (DSS) market leader in Europe. Read on
Business Objects in Top 10
AWARD In a top 100 data warehouse survey, DM Review readers have
Listo
Zona de Internet

informes y mensajes en lenguaje habitual, basados en condiciones o eventos de negocio inusuales.

Este sistema aporta además a los usuarios numerosas opciones para la distribución corporativa a otros usuarios en sus empresas, como pueden ser la multidifusiónn de informes, la edición, la difusión a los dispositivos, entre otros.

El servidor tiene un coste de mantenimiento muy bajo debido a que ha sido diseñado para atender las necesidades de un gran número de despliegues empresariales a gran escala, tanto dentro de una misma empresa como externos a ésta. Las principales ventajas que consiguen este bajo coste son por ejemplo el acceso desde full client y desde thin client, también que utiliza una sola infraestructura, o un único conjunto de herramientas de administración.

Además, a través de un acuerdo de partners tecnológico, BROADCAST SERVER proporciona notificación de eventos avanzada y servicios de respuesta para la empresa.

Si lo desea puede ampliar información sobre éste y otros productos de la compañía en la siguiente dirección www.businessobjects.com

INFORMATION BUILDERS COLABORA CON MICROSOFT EN EL LANZAMIENTO DE SQL SERVER 7.0

Information Builders participará en el lanzamiento de la iniciativa de data warehouse SQL Server 7.0 de Microsoft. Como parte del lanzamiento está distribuyendo demos en CD a 14.000 partners en todo el mundo. Este CD-ROM contiene demostraciones del SQL Server 7.0 y software de valor añadido, incluida la familia de herramientas de data warehouse SmartMart de Information Builders.

Estos componentes amplían la utilización, el rendimiento y el valor de SQL Server 7.0.

WebFOCUS el sistema de análisis y generación de informes para la Web corporativa, permite a los usuarios de SQL Server crear informes preparados para Web, uniendo vistas no sólo de la plataforma NT sino también de otras 30 más, incluidas UNIX y MVS.

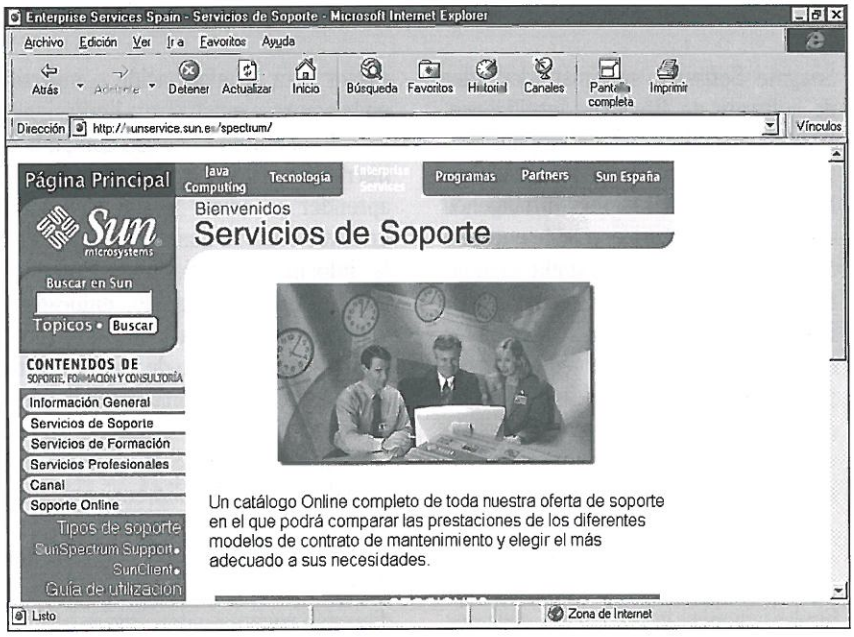
Todos estos sistemas explotan la estructura de acceso a datos común de Information Builders, proporcionando de este modo un potente acceso a cualquier base de datos o aplicación, en cualquier plataforma informática y trabajando con todos los diferentes sistemas y arquitecturas, que existen, en red.

La dirección a la que puede acceder para consultar en Internet es www.ibi.com

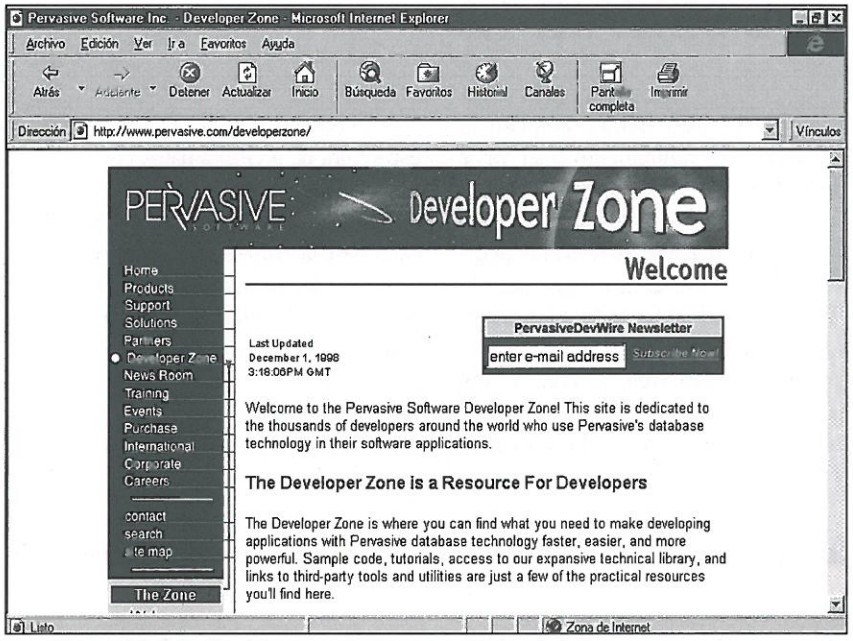
SUN TRABAJA CON LA COMUNIDAD DE LINUX PARA EXPORTAR EL JDK 1.2 A SU SISTEMA OPERATIVO.

Sun Microsystems anuncia su intención de brindar soporte a la comunidad de desarrolladores de Linux para exportar el Java Development Kit (JDK) 1.2 a Linux. El JDK es un conjunto de bloques básicos que incluye herramientas de desarrollo, una rica serie de bibliotecas de clases y una máquina virtual. Esta versión lleva incorporadas mejoras en estabilidad, rendimiento y compatibilidad

Sun certificará que el JDK para Linux es un lugar de total fiabilidad para los usuarios que desarrollen e implementen aplicaciones basadas en Java, servlets y Enterprise JavaBeans. Para ampliar la información puede consultar en la siguiente dirección www..sun.es



PERVASIVE SOFTWARE ANUNCIA PERVASIVE.SQL PARA WINDOWS CE


Pervasive Software Inc. - Developer Zone - Microsoft Internet Explorer
Archivo Edición Ver Ir a Favoritos Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Historial Canales Pant. completa Imprimir
Dirección http://www.pervasive.com/developerzone/
Vínculos
PERVASIVE SOFTWARE
Developer Zone
Welcome
Home
Products
Support
Solutions
Partners
Developer Zone
News Room
Training
Events
Purchase
International
Corporate
Careers
contact
search
site map
The Zone
PervasiveDevWire Newsletter
enter e-mail address
Subscribe Now!
Last Updated
December 1, 1998
3:18:06PM GMT
Welcome to the Pervasive Software Developer Zone! This site is dedicated to the thousands of developers around the world who use Pervasive's database technology in their software applications.
The Developer Zone is a Resource For Developers
The Developer Zone is where you can find what you need to make developing applications with Pervasive database technology faster, easier, and more powerful. Sample code, tutorials, access to our expansive technical library, and links to third-party tools and utilities are just a few of the practical resources you'll find here.
Listo
Zona de Internet

Pervasive.SQL para Windows CE es un motor de base de datos completo de altas prestaciones diseñado desde el principio específicamente para dispositivos de mano.

Su arquitectura proporciona a los desarrolladores la capacidad de elegir entre una gran variedad de componentes para integrarlos en sus aplicaciones compatibles con Windows CE, incluyendo además el replicado de datos, la gestión de eventos, la administración de catálogos, el proceso de solicitudes, la seguridad de acceso y los servicios web. Pueden obtener más información los desarrolladores en la siguiente página de internet www.pervasive. com/developerzone

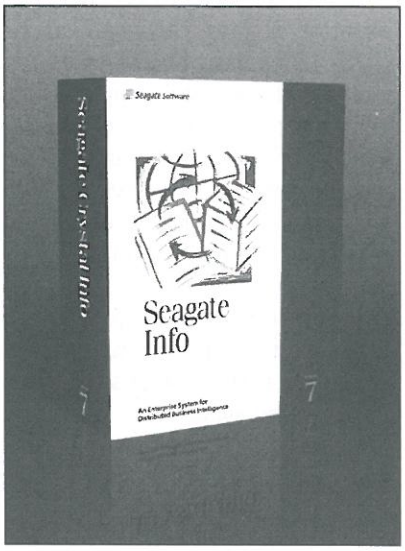
SEAGATE SOFTWARE PRESENTA LA SOLUCIÓN MÁS COMPLETA Y ESCABLE DE BUSINESS INTELLIGENCE DEL MERCADO

Seagate Software suministrador líder de software de Business Intelligence, ha presentado muy recientemente Seagate Info 7, una suite integrada de herramientas de Business Intelligence con una arquitectura escalable que permite a los usuarios, independientemente de su número, analizar y compartir información de forma fiable en la empresa. Antes conocido como Seagate Cristal Info, Seagate Info 7 es un producto de cuarta generación basado en la experiencia y en los requerimientos de los clientes.

LAs herramientas tradicionales de BI no están diseñadas para grandes implantaciones, y la demanda cada vez mayor para la información compartida en la empresa enfatiza sus limitaciones.

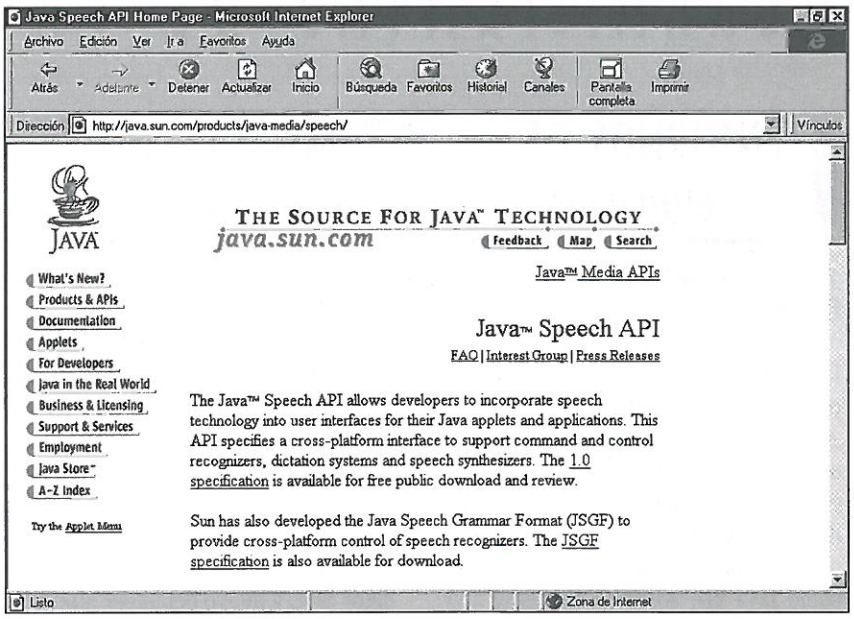
Esta Suite ofrece una completa de herramientas de análisis fáciles de aprender y utilizar y que ofrece una gran variedad de maneras de acceder a la información. Diseñado como un único paquete integrado, enfocado al usuario más que a la tecnología, incorpora todo el conjunto de acceso a datos y componentes de análisis incluyendo diseño de informes, consultas ad-hoc, edición avanzada en SQL, construcción OLAP, análisis multidimensional, etc..

Más información sobre los productos y servicios de Seagate Software en la dirección www.seagatesoftware.com



SUN ANUNCIA LA ESPECIFICACIÓN 1.0 DE JAVA SPEECH PARA SU PLATAFORMA

Ya está disponible, segun anunció Sun Microsystem, la especificación de Java Speech API 1.0, una interfaz multiplataforma independiente del fabricante que amplía la plataforma Java con el fin de permitir el reconocimiento y la síntesis de voz en applets y aplicaciones escritas en el lenguaje de programación Java.


Java Speech API Home Page - Microsoft Internet Explorer
Archivo Edición Ver Ir a Favoritos Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Historial Canales Pantalla completa Imprimir
Dirección http://java.sun.com/products/java-media/speech/
Vínculos
JAVA
THE SOURCE FOR JAVA™ TECHNOLOGY
java.sun.com
Feedback Map Search
What's New?
Products & APIs
Documentation
Applets
For Developers
Java in the Real World
Business & Licensing
Support & Services
Employment
Java Store
A-Z Index
Try the Applet Menu
Java™ Media APIs
Java™ Speech API
FAQ | Interest Group | Press Releases
The Java™ Speech API allows developers to incorporate speech technology into user interfaces for their Java applets and applications. This API specifies a cross-platform interface to support command and control recognizers, dictation systems and speech synthesizers. The 1.0 specification is available for free public download and review.
Sun has also developed the Java Speech Grammar Format (JSGF) to provide cross-platform control of speech recognizers. The JSGF specification is also available for download.
Listo
Zona de Internet

La API se caracteriza por una única interfaz para el desarrollo e implementación de una amplia gama de aplicaciones de voz tanto para ordenadores de sobremesa como para dispositivos portátiles y servidores de telefonía.

Java Speech API se ha desarrollado mediante un proceso de estándares abiertos, con significativas contribuciones de los productores líderes de tecnología Java, aplicaciones de voz y plataformas. Además permite a los desarrolladores la creación de aplicaciones de productividad personal, gestión de aplicaciones, interfaces compatibles con la tecnología de voz para usuarios discapacitados, mejoras en páginas web, dictado y corrección de pruebas. También puede utilizarse con las plataformas personalJava y EmbeddedJava para que el desarrollo de aplicaciones de voz se ajuste a dispositivos informáticos portátiles y de pequeño tamaño.

Junto a esta iniciativa, Sun anuncia la especificación 1.0 para el Java Speech Grammar Format.

Si desea ampliar información consulte en la web http://java.sun.com/products/javamedia/speech/

PLATINUM TECHNOLOGY PRESENTA UNA INNOVADORA SOLUCIÓN PARA LA INTERACCIÓN DE APLICACIONES Y BASES DE DATOS

La compañía Platinum Technology ha presentado una nueva solución PLATINUMSAL*Compass, una nueva solución que permite a los desarrolladores de aplicaciones visualizar y comprender la interacción entre aplicaciones y bases de datos, también reducirá el mantenimiento de la aplicación de bases de datos y eliminará la necesidad de volver a codificar manualmente las aplicaciones de las bases de datos.

Las aplicaciones de negocio se sustentan principalmente de la información en las bases de datos. Como consecuencia, cuando los desarrolladores realizan un cambio en la base de datos las aplicaciones pueden verse afectadas. Con esta nueva solución los desarrolladores ya no tienen que buscar manualmente el código de las aplicaciones que sufren el impacto de un cambio en las estructura de la base de datos.

Además de reducir el mantenimiento de las aplicaciones de las bases de datos también permite generar fácilmente componentes SQL reutilizables, eliminando la necesidad de volver a codificar manualmente dichas aplicaciones.

HERRAMIENTAS DE PROGRAMACIÓN

ENTORNOS DE DESARROLLO

JAVA DEVELOPMENT KIT 1.2 DE SUN

La ultimísima versión del entorno estándar de desarrollo en Java. Contiene todo el software y las herramientas necesarias para compilar, testear y ejecutar applets y aplicaciones desarrolladas en lenguaje Java. La versión 1.2 proporciona nueva funcionalidades, como Java 2D, accesibilidad, drag and drop, colecciones, métodos de introducción de código, identificación, IDL, JVMDI, y muchas más. Tanto es software como la documentación es totalmente libre de uso.

MULTIMEDIA BUILDER 3.1

Pensada para desarrollar aplicaciones profesionales sin necesidad de programar. Dispone de un Portapales propio que aumenta la potencia de acciones de reutilización de código gracias al Copiar y Pegar. Se pueden integrar textos, gráficos, sonidos y objetos.

PERL DEVELOPMENT KIT 1.1 BETA 2

Incluye un debbuger, un compilador, una librería DCOM y un creador de controles. Este kit de desarrollo está compuesto por cuatro componentes: PERL debugger es un ligero pero excelente debugger. La utilidad PL2EXE transforma Perl scripts en ejecutables que pueden, opcionalmente, ser ejecutados en ordenadores sin PERL. PerlCOM junto con DCOM, permite distribuir aplicaciones PERL fácilmente. PerlCtrl es el encargado la creación de controles escritos en PERL. Es de uso libre.

POWER BUILDER 6.0

Entorno especializado en la creación de aplicaciones de misión crítica. Es un entorno de desarrollo visual que proporciona la capacidad que se necesita para construir y desplegar aplicaciones web y distribuidas, con la misma facilidad de los desarrollos cliente/servidor. Esta versión incorpora nuevas herramientas para la creación de aplicaciones Web, acceso rápido a bases de datos SQL y nuevo debugger que analiza a la vez que se desarrolla.

D RAD 1.5

D Rad es un potente entorno para el desarrollo de juegos 3D. D Rad 3D es un potente paquete de utilidades para la creación de juegos 3D. Es muy sencillo de utilizar. Se pueden construir mundos interactivos en 3D en los que ubicar elementos animados inteligentes en 3D dentro de un espacio virtual.

CATALYST 3.15.01

Entorno para la creación de aplicaciones de procesamiento de datos. Es una herramienta de desarrollo de sistema diseñado para la rápida creación portátil de aplicaciones de procesamiento de datos. Está destinado para aplicaciones en desarrollo que usan tecnología de base de datos cliente-servidor.

GKSETUP 1.62

GUI que permite crear potentes instalaciones de programas. Soporta sistemas autoextraibles, varios discos, desinstalaciones, copia de librerías .DLL compartidas y en uso, registro de OCX, modificaciones del Menú Inicio, cambios en el Registro.

LENGUAJES

VISUAL BASIC

ACTIVEREPORTS 1.08

Creador de informes para Visual Basic. Se integra en el entorno de programación de Visual Basic. Permite utilizar un código VB, ActiveX y OLE en los informes, haciendo lo posible para generar practicamente todos los tipos de informes.

CALENDAR WIDGETS 1.0F

Controles de Visual Basic y ActiveX con funciones horarias. Conjunto de controles ActiveX y de Visual Basic para diseñar aplicaciones que incorporan funciones de gestión de fechas y horas. Además, pueden ser utilizados en la creación de programas de gestión de información personal como gestores de proyectos, controles de cuentas, facturación, etc.

21HEX DOCKIT 1.10.14A

Formularios y plantillas ajustables en Visual Basic 5.0. Es un control ActiveX que proporciona formularios o plantillas ajustables de Visual Basic. Una plantilla "docked"es aquella que está justo al margen de la ventana principal de la aplicación. DockIt permite realizar formularios en los que se pueden definir qué márgenes de la ventana, que la ventana

puede aparecer flotante, definir el estilo de la barra de títulos, fijar el tamaño y colocación de cada ventana, y más.

TIMETOWIN 6.19

900 funciones y subrutinas para Visual Basic 5.0. Es una colección de alrededor de 900 funciones y subrutinas para programadores de Visual Basic. Las DLL incluyen rutinas en las siguientes categorías: geometría, arrays, bitmaps, compresión, fecha y hora, encriptación, ficheros, cadena, control de lenguaje, list box, y más.

VBASSIST 5.02

15 herramientas para potenciar el entorno de Visual Basic 5.0. Entre ellas pueden encontrarse asistentes para: acelerar el teclado, un histórico del portapapeles, código, comentarios, creación de cuadros de diálogo, alineación de elementos, gestión de arrays, visualizar proyectos, recursos, zoom, etc.

JAVA

KAWA 3.13

Entorno de Desarrollo Integrado para crear aplicaciones Java. Proporciona un entorno excelente para desarrollar aplicaciones Java. Utiliza un gestor de proyectos visual junto con un debugger GUI, un navegador de clases, un browser de código, y editor de clases.

1 COOL BUTTON TOOL 4.0

Creación de botones Web con Java. El programa permite crear botones Java con numerosos estilos y comportamientos. Los atributos de los botones incluyen color, profundidad, transparecia, texto en color, bordes en color, sonidos AU y WAV, bitmaps JPEG y GIF, e incluso botones drag y drop.

EVANS JAVA TOOLKIT 1.0.1

Permite añadir verdaderas funcionalidades GUI en los applets Java. Incluye una herramienta de formularios para añadir características GUI en los frames Java, una clase MsgBox para la creación de cuadros de diálogo flotantes y una clase ClientIO para escribir en el servidor Web.

APPLET NAVIGATION FACTORY 1.0

Desarrollo de menús basados en Java para Internet. Permite crear menús basados en Java para facilitar la navegación en Web. Los menús pueden incorporar imagen y texto. Permite configurar los menús con gran variedad de estilos de bordes y diversos colores.

SMARTSCRIPT 5.0 BETA 1

Genera sencillamente toda variedad de scripts para JavaScript. Tan sólo rellenando los espacios en blanco puede generar más de 15 JavaScripts con efectos distintos para la Web. Se pueden crear botones para dirigir el ratón, abrir ventanas, detectar el navegador utilizado, crear textos, mostrar un reloj con la hora real, visualizar la fecha, y más.

JAVASCRIPT BUTTONS II 1.5

Botones de alta calidad con JavaScript
Genera el código JavaScript para poner una imagen en una página Web y que cambie cuando el ratón pase sobre ella. Esto se usa comúnmente para generar botones que cambian su apariencia cuando el cursor del ratón pasa sobre ellos. El código JavaScript que crea puede ser copiado y pegado en archivos HTML.

HTML

HOTMETAL PRO 5.0

Sin duda una de las mejores herramientas de autor de creación Web. Dispone de varias vistas del documento, asistentes y plantillas para todas las funciones. Incluye las últimas novedades de HTML, DHTML y XML. El paquete incluye además una herramienta para gestión de sites completos, otra de edición de imágenes, un servidor Web, y una utilidad de validación de código HTML.

MACROMEDIA DREAMWEAVER 2.0

La ultimísima versión de la herramienta más usada por los profesionales. Es una herramienta especializada en la creación de contenidos dinámicos y permite a los diseñadores crear sofisticadas plantillas y animaciones sin utilizar una línea de código.

BATCH IT! 1.2B

Procesamiento rápido de operaciones con elementos Web. Es una utilidad de procesamiento batch que permite cambiar el tamaño automáticamente, convirtiendo, renombrando y aplicando filtros a imágenes para la Web.

COOL PAGE 1.1

Es una herramienta visual para el diseño de código HTML de principiantes. Crea páginas Web simplemente con copiar y pegar, y cambiar las propiedades de los objetos.

EAUTHOR SITE 3.04

Interesante herramienta de autor basada en plantillas. Es una herramienta de autor, basada en plantillas, especial para desarrolladores de grandes sites Web. Incluye un potente editor WYSIWYG, un editor de código HTML, un gestor de proyectos visual, un modo de previsualización instantánea, y diferentes propiedades para autores Web.

HELP & MANUAL 1.4H

Herramienta de autor Web que también genera manuales técnicos. Es una herramienta de autor WYSIWYG que puede ser usada para generar manuales impresos de aspecto profesional y para crear páginas HTML. . Permite crear y desarrollar fácilmente el contenido de la ayuda incluyendo las habituales imágenes, botones, macros, objetos OLE, y links. Usando los mismos datos, también

permite crear páginas HTML, manuales impresos y archivos RTF.

WEBCONVERT 2.02

Transforma documentos de procesadores de texto en páginas HTML. Permite añadir acciones en la Web con una variedad de plantillas y estilos que incluyen imágenes, colores, encabezados, pies de página, cuerpo del texto, hipertexto, iconos, márgenes, y más.

OTROS

ACTIVETHREED PLUS 2.0

Controles ActiveX para potenciar aplicaciones Web. ActiveThreed Plus es un conjunto de once controles ActiveX que proporcionan a las Web el aspecto de las modernas aplicaciones de Internet. Incluye: un sencillo creador de frames, un sencillo creador de animaciones con imágenes.

ALIB 4.0

Gran librería dedicada a funciones de lenguaje Ensamblador. Programa freeware que incluye una gran librería dedicada a funciones del lenguaje Ensamblador. Alrededor de 400 funciones diferentes, la mayoría con sus fuentes de códigos, están incluidas en este programa.

BCB BACKUP 1.1

Copias de seguridad para usuarios de Borland C++ Builder. BCB Backup es una utilidad de copia de seguridad para usuarios de Borland C++ Builder. El programa proporciona un camino simple para desarrollar una copia seguridad de los fuentes, cabeceras y ficheros de proyecto.

CLASSBUILDER 2.0

Freeware que permite acceder y manipular clases o categorías C++. Permite crear, manipular y gestionar clases así como establecer relaciones entre ellas a través de un simple interfaz gráfico de usuario. Puede generar código para definir relaciones y que sus asociados encuentren métodos, insertar códigos, sincronizar códigos con entornos designados.

IRIE PASCAL 1.00

Compilador e intérprete de Pascal. Tanto el compilador como el intérprete están basados en la línea de comandos. El compilador no compila el código nativo (es decir, no genera instrucciones para un ordenador real), en cambio genera instrucciones (también llamadas Códigos Byte) que pueden ser ejecutadas por el intérprete ("Irie Virtual Machine")

HERRAMIENTAS Y UTILIDADES

EDITPLUS 1.21

Editor de textos con funciones específicas para desarrolladores. Es un editor de texto con un potente paquete de características para editores y programadores de Internet. Ofrece un gran surtido de características, incluyendo: capacidad para ir subrayando sintaxis (para HTML, ASP, JavaScript, VBScript, PERL, Java, C/C++, y muchos otros).

LINUX-IN-DOS 0.9

Utilidad para leer y convertir ficheros Linux a Windows. Permite a los usuarios leer ficheros del sistema Linux y copiar esos ficheros directamente en formato Windows. Además puede guardar ficheros Linux con formato Windows.

NET VAMPIRE 3.01

Descarga automatizada de ficheros a través de Internet. Interesantes opciones de programación e integración con el navegador.

ULTRAEDIT PROFESSIONAL TEXT/HEX EDITOR 6.00A

Es un potente y excelente editor de texto para programadores. Incluye múltiples ventanas, ilimitados tamaños de ficheros, potente búsqueda y sustitución (en todos los ficheros o todos los ficheros abiertos), multinivel de deshacer, barra de herramientas configurable, inserción de fecha y hora, potente soporte para macros, línea automática de indexación.

WEBZIP 2.50

Permite guardar páginas Web en un fichero ZIP comprimido. WebZips descarga páginas Web completas y las ubica dentro de un archivo con formato ZIP para con posterioridad visualizar las páginas o trabajar con ellas en modo off-line.

WINLOADER 4.02

Utilidad freeware que permite controlar los procesos de arranque de Windows 95/98. WinLoader proporciona el control total sobre aplicaciones que son lanzadas cuando carga Windows 95/98.

REDES

LOCALES

IMAGECAST LE 1.0

Utilidad que permite clonar equipos conectados a una red. Es una utilidad multicasting para la creación de imágenes de discos que permite clonar PC´s de una red. Utilizando la imagen fabricada para el disco de algunos sistemas, se pueden crear tantos clónicos como se necesiten.

REMOTE TASK MANAGER 1.3.3

Manejar o dirigir tareas y procesos sobre otros ordenadores NT. Permite

ver directamente todas las tareas y procesos en ordenadores remotos y visualizar información acerca de las mismas así como finalizarlas.

DISTRIBUIDAS

@MAILGATE 2.59

Sistema servidor de procesamiento de correo electrónico para redes. Hace de puente entre el servidor POP3 del ISP y la red local. Permite compartir una cuenta de E-mail con varios usuarios de un grupo que trabaje con MS Mail o cc:Mail.

NETPROXY 3.02

Proxy y Firewall para LAN. Es un servidor proxy y un firewall que permite que todos los de usuarios de una LAN accedan a Internet. Puede ser vía SLIP, PPP, o TCP/IP. Ya que la mayoría de aplicaciones de Internet utilizan el TCP/IP para conectarse a Internet, se debe instalar TCP/IP en cada una de las máquinas de la LAN desde las que se desee acceder, y dar a cada máquina una IP propia.

WINGATE 2.1D PARA NT

Permite conectar un entorno LAN a Internet utilizando una única conexión. Puede trabajar con su navegador favorito, E-mail, noticias, FTP, Telnet, y más, sin el coste de tener que instalar un router, o instalando líneas de teléfono adicionales. WinGate también trabaja con el Proveedor de Servicios Internet, quien asigna números IP aleatorios.

OTROS

NETWORKCHARTER/PRO 3.0

Excelente herramienta para el diseño de redes para profesionales. Esta aplicación está exclusivamente indicada para los profesionales de la informática que realizan el mantenimiento, diseño y configuración de redes, sean del tamaño que sean. Para facilitar la labor del usuario incluye una base de datos orientada a objetos y varias herramientas que combinan las funciones de diseño y documentación de una red. Este programa necesita este número de serie para su correcta instalación: 100000 2302541996 6686.

VISUALROUTE 4.0A

Traceador que analiza los problemas de conectividad. Es una rápida utilidad traceadora que analiza problemas relacionados con la conectividad.

XITAMI WEB SERVER 2.4B BETA

Un rápido servidor Web multi-threaded. Carece de algunas de las características encontradas en muchos servidores Web disponibles en el mercado. No obstante, compensa en rapidez y habilidad para manejar grandes volúmenes de tráfico de información. Soporta las principales operaciones HTTP.

WAR FTP DAEMON 1.70 BETA 1 R4

Excelente servidor FTP totalmente libre. Servidor de FTP completamente gratuito que incluye gran facilidad de configuración y potentes opciones de seguridad.

DOCUMENTACIÓN Y TUTORIALES

BUGCOLLECTOR PRO 2.0

Base de datos para registrar y organizar bugs. BugCollector Pro es una base de datos para rastrear bugs, así como para poder ayudar a manejar los pequeños detalles finales de los proyectos de desarrollo de software. También puede organizar los errore de diferentes modos como por estados, prioridades, intensidad y manejar o preparar las notas asociadas con cada fallo.

INDEX DOT HTML

Referencia técnica para el estudio a fondo de lenguajes HTML y CSS. Incluye información sobre las etiquetas más populares y estándares, así como de los browsers o navegadores que las soportan.

WINDOWS 95 REGISTRY, THE

Presentación de información técnica sobre el Registro de Windows. Son los extractos de una conferencia ofrecida por Mark Russinovich para el Grupo de Usuarios de Windows 95 del ya desaparecido Boston Computer Society (BCS), el 23 de Septiembre de 1996.

TABLE TUTOR 2.01

Guía de sencillo manejo para hacer tablas HTML para documentos Web. Presentada en formato HTML, conducirá al usuario, paso a paso, a través de los principales tags de las tablas y su comportamiento en la Web.

FUENTES DE LOS ARTÍCULOS

Glide (II)
Los fuentes de la práctica propuesta en el capítulo de este mes.

HTML Dinámico (III)
Ejemplos de algunos de los efectos realizados con HTML Dinámico.

Reconocimiento de voz (VI)
Fuentes de la práctica propuesta en el artículo.

Planificación de procesos en Linux
Entregamos los fuentes de la práctica realizada en el artículo.

Creación de un Chat con Java
Los fuentes del ejemplo práctico propuesto en el artículo.

Áreas desplegables con HTML Dinámico (III)

Adolfo Aladro (aaladro@arrakis.es)

Continuando con nuestra serie de artículos entorno al HTML Dinámico vamos a profundizar en todos los conocimientos adquiridos hasta el momento, al mismo tiempo que creamos un nuevo y útil efecto. Partiremos de la librería que hemos ido creando a lo largo de esta serie para sacarle todo el jugo posible.

ÁREAS DESPLEGABLES

¿Qué ocurre cuando tenemos un documento demasiado largo y queremos publicarlo en la Web? Está claro que la solución más obvia, poner todo el documento en un sola página *HTML*, no es un salida muy elegante. También está comprobado que cuando un usuario se topa con una página demasiado larga termina por ignorar el final de la misma. Por este motivo normalmente se fracciona el documento en varias páginas *HTML* y se crea un índice, que bien puede estar en un marco o en diferentes páginas.

Aunque esta solución resulta bastante acertada, el *HTML* Dinámico ofrece la posibilidad de poder colocar todo el documento en una sola página *HTML* de una manera amigable y atractiva para el usuario. Se trata de crear un índice de forma que al hacer clic sobre una entrada se muestre su contenido dentro de la misma página.

Cuando el usuario haga clic sobre otra entrada diferente del índice, el contenido de la anterior desaparecerá y aparecerá el de la nueva. La Figura 1 muestra un esquema del funcionamiento de este efecto.

DIVIDIENDO EL DOCUMENTO EN CAPAS

Para conseguir nuestro propósito dividiremos nuestro documento en varias capas, que en este caso serán de dos tipos: las capas "marcador", que

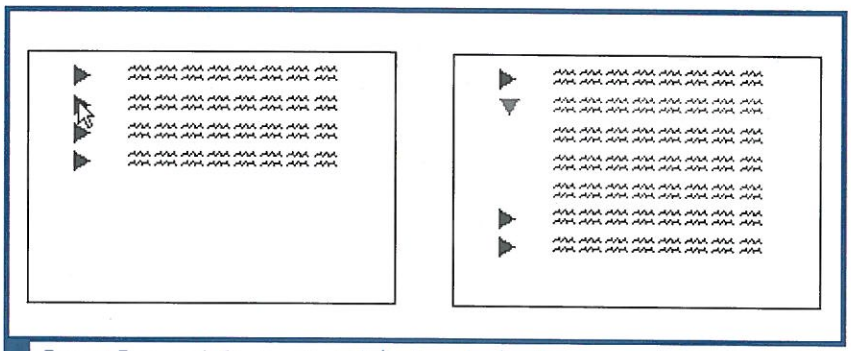


Figura 1. Esquema de funcionamiento de las áreas desplegables.

contienen las entradas del índice sobre las que el usuario hará clic para ver los contenidos asociados y las capas "contenido" donde se almacenarán los contenidos vinculados con cada entrada del índice. Una vez que tenemos configurada la estructura lógica, el resto del trabajo consiste, a grandes rasgos, en mostrar, ocultar y mover todas las capas de forma adecuada cada vez que se produce un clic sobre alguna capa marcador. Veamos paso a paso cómo debemos efectuar estas tareas.

Listado 1. Definición del arrray que contiene el texto de cada marcador.

```
var marcadores = new Array();

marcadores[0] = "Like A Prayer";
marcadores[1] = "I´m Breathless";
marcadores[2] = "The Immaculate Collection";
marcadores[3] = "Erotica";
marcadores[4] = "Bedtime Stories";
marcadores[5] = "Something To Remember";
marcadores[6] = "Evita";
```

LAS CAPAS "MARCADOR"

Las capas marcador contendrán las entradas del índice correspondiente. Básicamente están formadas por un texto y un icono que indicará si su contenido asociado está desplegado o no. Es inmediato observar que todas las capas marcador guardan características similares tanto en lo relativo a su aspecto como en lo que concierne a su funcionalidad.

La función "iniciar" es la encargada de inicializar todos los parámetros necesarios para que el script funcione

Por este motivo vamos a definir una función que a partir de un parámetro con el número del marcador se encargue de generar el código *HTML* asociada a ese marcador. El texto de cada marcador se guardará en un *array* y la función lo tomará de ahí para crear el código *HTML* adecuado en cada caso. El listado 1 muestra la definición de dicho *array* que contendrá el texto de cada marcador y el listado 2 muestra la función encargada de generar el código *HTML* de cada marcador. Finalmente el listado 3 contiene un ejemplo de cómo quedarán los marcadores en la página *HTML*. Como podemos apreciar, a medida que se va cargando la página se generará el código *HTML* de cada capa "marcador".

LAS CAPAS "CONTENIDO"

Cada capa contenido albergará los contenidos asociados al marcador correspondiente. Puesto que ese contenido dependerá de cada caso, aquí no conviene crear ningún tipo de función *JavaScript* que genere el código *HTML* correspondiente tal y como hemos hecho con las capas marcador.

LA FUNCIÓN "INICIAR"

La función iniciar es la encargada de inicializar todos los parámetros necesarios para que el script funcione. Cuando el documento se carga, momento éste que viene determinado por el evento *onload*, pasamos a crear los objetos correspondientes, los colocamos en su lugar adecuado y en definitiva dejamos todo preparado para que la interacción con el usuario pueda llevarse a cabo de manera correcta.

Para la conveniente colocación de los elementos en el documento es imprescindible conocer el tamaño de la ventana disponible. Así, lo primero que hacemos al comienzo de la función es asignar a unas variables globales los valores correspondientes, tal y como se aprecia en el listado 4. A partir de este instante la variable *ancho_ventana* contendrá el ancho de la ventana tanto en NS4 como en IE4, y lo mismo ocurrirá con la variable *alto_ventana*.

Posteriormente procedemos a colocar todos los elementos (Ver listado 5). Para que esta labor pueda realizarse de manera automática hemos llamado a las capas marcador *"marcador0"* ··· *"marcadorn"* y a las capas contenido *"segmento0"* ··· *"segmenton"*. De esta forma es posible iniciar todos los valores dentro de un bucle *for* haciendo uso de la función *eval*.

Las capas "marcador" contendrán las entradas del índice correspondiente

La manera en la que se calculan las posiciones resulta bastante obvia. La posición de un determinado marcador vendrá dada por la suma de la posición de la capa anterior, su longitud y un espacio de separación que fijamos a través de la variable global *separador*. Evidentemente las capas contenido seguirán el mismo esquema. Así la posición de una capa "contenido" vendrá determinada por la suma de la posición del marcador que

Listado 2. Función que genera el código HTML asociado.

```
function escribirMarcador(que_marcador) {
  codHTML = '<TABLE>';
  codHTML += '<TR>';
  codHTML += '<TD VALIGN="MIDDLE">';
  codHTML += '<A HREF="#" NAME="#" onmouseover="self.status=\'' + marcadores[que_marcador] + '\'; return true" onmouseout="self.status=\'\'; return true" onclick="return arreglar(' + que_marcador + ')"><IMG NAME="img_marcador' + que_marcador + '" SRC="apuntador_right.gif" WIDTH="20" HEIGHT="20" BORDER=0></A>';
  codHTML += '</TD>';
  codHTML += '<TD VALIGN="MIDDLE">';
  codHTML += '<A HREF="#" NAME="#" onmouseover="self.status=\'' + marcadores[que_marcador] + '\'; return true" onmouseout="self.status=\'\'; return true" onclick="return arreglar(' + que_marcador + ')">';
  codHTML += marcadores[que_marcador] + '</A></TD>';
  codHTML += '</TR>';
  codHTML += '</TABLE>';

  return codHTML;
}
```

lleva asociado, su longitud y el espacio separador. No obstante, nada más cargarse la página sólo se mostrarán los marcadores, con lo que en la función iniciar no se calculan los valores correspondientes a la posición de cada capa contenido. En resumen, la función inicializar hace para cada capa:

Posición de la capai = Posición de la capai-1 + Longitud de la capai-1 + Margen de separación

La función "arreglar" se encarga de manejar eventos onclick que reciban los marcadores

Nótese también que a lo largo del bucle *for* asignamos un valor a la propiedades *width* (ancho de la capa) y *left* (posición horizontal) de las capas sólo en el caso de que el navegador sea *IE4*. Como ya apuntamos en nuestro artículo anterior, esto se debe a las diferencias existentes entre uno y otro *IE4* y *NS4* a la hora de proporcionarnos los valores correspondientes al ancho de la ventana y a la hora de manejar la propiedad width de una capa. El ancho disponible de la ventana lo obtenemos dependiendo del navegador de la siguiente forma:

IE4 document.body.clientWidth

NS4 window.innerWidth

La diferencia que existe entre estas dos propiedades es que *NS4* basa su propiedad en la ventana, la cual existe antes de que la página se cargue, mientras que *IE4* basa su propiedad en el cuerpo de la página, y este valor no lo conocemos hasta que la página ha sido cargada. Por otro lado, *NS4* necesita conocer el valor del atributo *width* de un elemento antes de que éste sea creado, es decir, debe aparecer en la definición del estilo. Con *IE4* podemos asignar un valor a la propiedad *width* en cualquier momento.

La posición de una capa "contenido" vendrá determinada por la suma de la posición del marcador que lleva asociado, su longitud y el espacio separador

LA FUNCIÓN "ARREGLAR"

Esta función se encargará de manejar los eventos *onclick* que reciban los marcadores, bien sea sobre su icono o sobre el texto en sí (Si nos fijamos en el listado 2 podemos observar que tanto el icono como el texto de cada marcador es un enlace susceptible de recibir eventos de este tipo). Su tarea consistirá en recalcular las posiciones de todos los elementos teniendo en cuenta:

Listado 3. Ejemplo de los marcadores en la página HTML.

```
<DIV ID="marcador0" CLASS="estilocapas">
<SCRIPT LANGUAGE="JavaScript">
<!—
 document.write(escribirMarcador(0));
//—>
</SCRIPT>
</DIV>
```

Listado 4. Ancho y alto de la ventana del navegador.

```
if (NS4) {
    ancho_ventana = window.innerWidth;
    alto_ventana = window.innerHeight;
  } else {
    ancho_ventana = document.body.clientWidth;
    alto_ventana = document.body.clientHeight;
  }
```

1. Si no hay ningún contenido mostrándose, es decir, sólo se ven los marcadores, debe "abrir" el marcador correspondiente desplegando así su contenido.

2. Si ya hay un marcador "abierto", debe "cerrarlo" (i.e. debe ocultar su contenido) y mostrar el contenido del marcador sobre el cual se ha hecho clic.

3. Si el marcador sobre el que se hace clic estaba "abierto" simplemente lo cierra.

Es conveniente que volvamos a hacer hincapié en el uso de la función "eval"

Este es el esquema de funcionamiento que hemos establecido aunque podríamos haber fijado cualquier otro, como por ejemplo que se permitieran tener todos los marcadores que se quisiera abiertos, o cualquier otra combinación. En líneas generales la programación sería similar y sólo habría que controlar qué capas hay que mostrar u ocultar en cada momento, calculando en consecuencia la posición de las mismas. Para controlar en qué estado nos hayamos de los tres descritos anteriormente hemos definido una variable global llamada *que_segmento_expandido*.

Su valor será igual a –1 si todos los marcadores están cerrados, y en otro caso contendrá el número del marcador que se encuentra desplegado en ese momento. Si el número de marcador sobre el que se ha hecho clic coincide con el valor de *que_segmento_expandido* nos hallamos en el tercer supuesto (Estamos cerrando un marcador abierto). Si no es así, estamos en el segundo caso (Estamos abriendo un nuevo marcador, lo que implicará que se cierre el que estaba abierto previamente).

Además de lo anterior, esta función deberá cambiar los iconos correspondientes que indican si un marcador está "abierto" o "cerrado". Para llevar a cabo esta labor se ha asignado al atributo *NAME* correspondiente a la imagen de cada icono el nombre *img_marcadorn*, donde *n* es el número de marcador.

Hay que tener en cuenta de nuevo las diferencias entre el modelo de objetos de *IE4* y *NS4*. Mientras el primero permite acceder a cualquier imagen del documento, esté o no dentro de una capa, para el segundo una imagen pertenece al objeto *document* de la capa que la contiene. De ahí que tengamos que acceder de forma distinta a cada objeto imagen para cambiarlas si queremos cambiar el icono del marcador. A continuación se describen ambas alternativas:

IE4 eval('document.images["img_ marcador' +i + '"].src = "apuntador_ down.gif";');
NS4 eval('elmarcador' + i + '.document .images["img_marcador' + i + '"].src = "apuntador_down .gif";');

Si la función "arreglar" devolviera "true" el navegador intentaría ir a la dirección enlace

Es conveniente que volvamos a hacer hincapié en el uso de la función *eval*. Esta función se ha utilizado con gran porfusión a lo largo de esta serie de artículos pero quizá este es uno de los casos en donde mejor se revela su gran potencia y utilidad. Si no tuviéramos la función *eval* sería necesario recorrer una a una todas las capas "marcador", todas

Listado 5. JavaScript encargado de colocar las capas "marcador".

```
for(i=0;i<numero_segmentos;i++) {
  eval('elmarcador' + i + ' = new objetoCapa("marcador' + i + '");');
  if (!NS4) {
    eval('elmarcador' + i + '.width = ancho;');
  }
  eval('elmarcador' + i + '.left = a;');
  eval('elmarcador' + i + '.top = posY;');
  eval('elmarcador' + i + '.mostrar();');
  eval('elsegmento' + i + ' = new objetoCapa("segmento' + i + '");');
  if (!NS4) {
    eval('elsegmento' + i + '.width = ancho;');
  }
  eval('elsegmento' + i + '.left = a + indentacion_segmento;');
  eval('posY += separador + elmarcador' + i + '.getHeight();');
```

las capas "contenido" viendo sus valores, o por ejemplo, tendríamos que inundar el código de sentencias de tipo *if-then-else* a la hora de planear los cambios de icono.

```
if (marcador == 1) {
  "Cambiar la imagen del marcador 1"
} else if (marcador == 2) {
  "Cambiar la imagen del marcador 2"
} else if (marcador == 3) {
  "Cambiar la imagen del marcador 3"
}
...
}
```

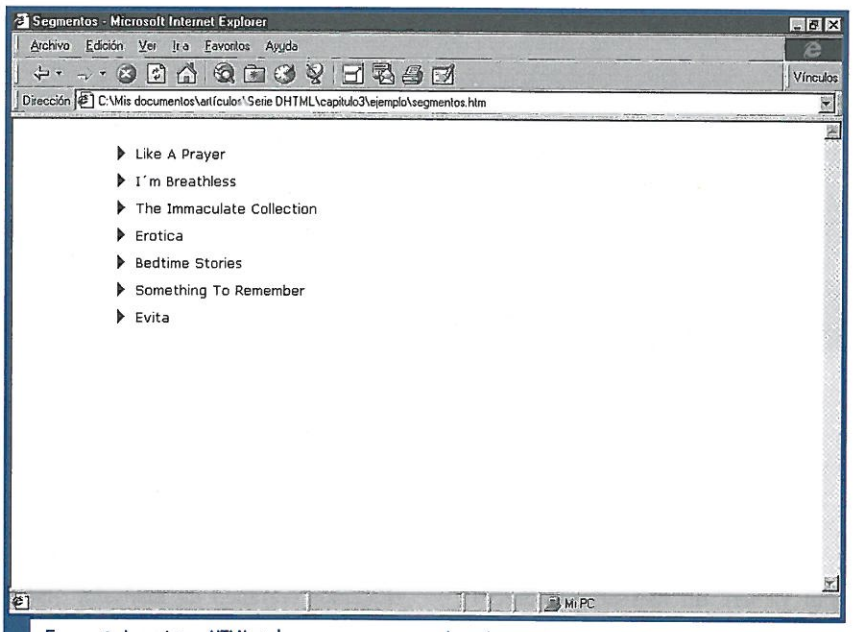


Figura 2. La página HTML tal y como aparece nada más cargarse.

ENLACES A NINGUNA PARTE

Finalmente vamos a comentar un pequeño detalle que no por ser pequeño carece de importancia. Se trata de un aspecto que ya se había utilizado, pero que hasta el momento no hemos explicado convenientemente. Se trata de un pequeño truco que puede resultar de gran utilidad, especialmente en el desarrollo de páginas Web en las que se hace uso de *JavaScript*. Se trata de los enlaces a ninguna parte, es decir, aquellos enlaces que son capaces de recibir clics pero que no nos llevan a ninguna parte.

Quizás el lector se haya preguntado tras haber inspeccionado el código fuente de nuestro ejemplo, por qué la función *arreglar* devuelve siempre **false**. Bien, si volvemos sobre el listado 2 podemos ver que la llamada a esta función aparece en los enlaces de la siguiente forma:

```
<A NAME="#" HREF="#" onclick="return
    arreglar()">···</A>
```

Si la función *arreglar* devolviera **true** el navegador intentaría ir a la dirección del enlace. Como nuestra función *arreglar* devuelve siempre el valor **false**, el navegador anula la ejecución de ese enlace siempre. En definitiva, tenemos un elemento susceptible de recibir clics pero que no nos lleva a ninguna parte.

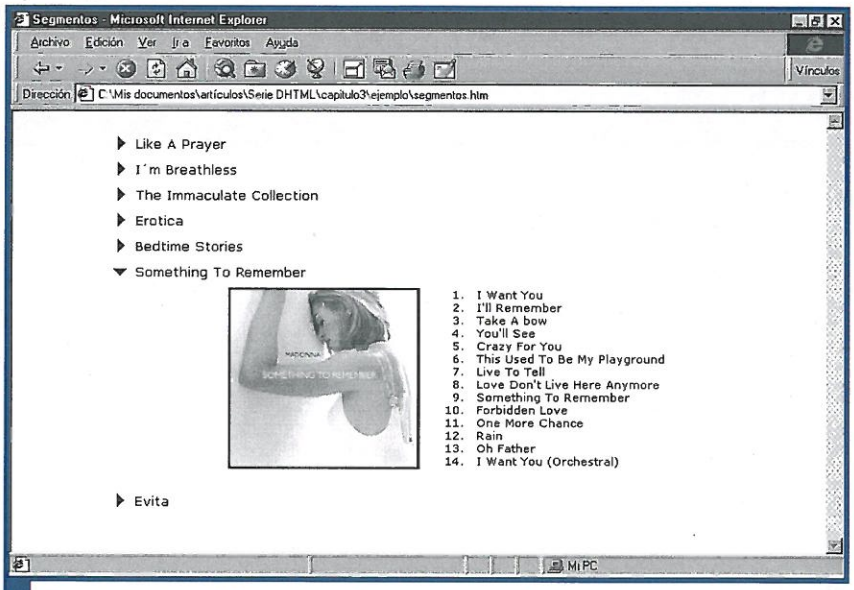


Figura 3. La página HTML tras hacer clic sobre un marcador.

El texto de cada marcador se guardará en un array y la función lo tomará de ahí para crear el código HTML más adecuado

Las figura 2 muestra la página HTML de nuestro ejemplo tal y como aparece inmediatamente después de cargarse. La Figura 3 muestra la página tras haber hecho clic sobre un marcador. Como vemos se trata de un sencillo ejemplo, pero con recursos valiosos que podremos utilizar en multitud de ocasiones.

Reconocimiento de voz (VI)

Constantino Sánchez Ballesteros (constantino@nexo.es)

Como prometimos el mes pasado, en este nuevo artículo vamos a crear un programa que se encargará de recoger los E-mails que tengamos almacenados en nuestro servidor y a continuación nos leerá el asunto del que tratan los mismos de una forma automática.

INTRODUCCIÓN

Puesto que vamos a trabajar en este proyecto con varios controles para mensajería, daremos unos datos para los neófitos en el uso de este tipo de controles. Los controles de la Interfaz de programación de aplicaciones de mensajería (*MAPI*) permiten crear aplicaciones *MAPI* de Visual Basic con correo electrónico. Hay dos controles de este tipo:

- *MAPISession*
- *MAPIMessages*

El control *MAPISession* inicia una sesión y establece una sesión *MAPI*, además de utilizarse en el momento de terminar una sesión de este tipo. El control *MAPISession* permite al usuario realizar diversas funciones del sistema de mensajería.

Se trata de controles son invisibles en tiempo de ejecución, que se caracterizan por la ausencia de eventos. De esta forma, para utilizarlos se deben especificar los métodos adecuados. Después de establecer una sesión de mensajería con el control *MAPISession*, el control *MAPIMessages* se encarga de efectuar las funciones requeridas para gestionar el sistema de mensajería.

Para que los controles funcionen deben estar presentes los servicios *MAPI*, los cuales se proporcionan con los sistemas que permiten correo electrónico. Si intentamos ejecutar un programa que utilice los controles *MAPI*, debemos

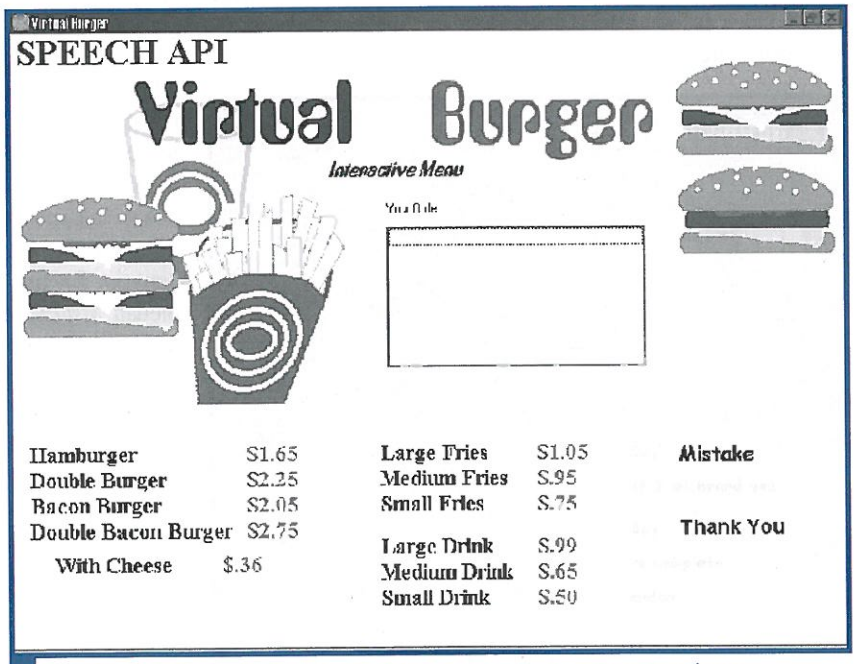


Figura 1. Esta aplicación incluida en el SDK gestiona una hamburguesería con la voz.

asegurarnos de que tenemos las *DLL* de *MAPI* de 32 bits instaladas correctamente, ya que de lo contrario no podremos realizar funciones sencillas de *MAPI* como SignOn. Por ejemplo, en Windows 95 debemos instalar *Mail* durante el proceso de instalación del sistema operativo o realizar después la instalación desde el *Panel de control* para poder usar desde Visual Basic las funciones de *MAPI* o los controles personalizados.

Con el control *MAPIMessages* podemos:

- Tener acceso a los mensajes actuales de la Bandeja de entrada.
- Redactar un nuevo mensaje.
- Agregar y eliminar destinatarios de mensajes y datos adjuntos.
- Enviar mensajes (con o sin interfaz de usuario compatible).
- Guardar, copiar y eliminar mensajes.
- Mostrar el cuadro de diálogo *Libreta de direcciones*.
- Mostrar el cuadro diálogo *Detalles*.
- Tener acceso a datos adjuntos, incluyendo los de vinculación e incrustación de objetos (*OLE*).
- Resolver un nombre de destinatario durante el direccionamiento.
- Efectuar acciones tales como responder, responder a todos y reenviar mensajes.

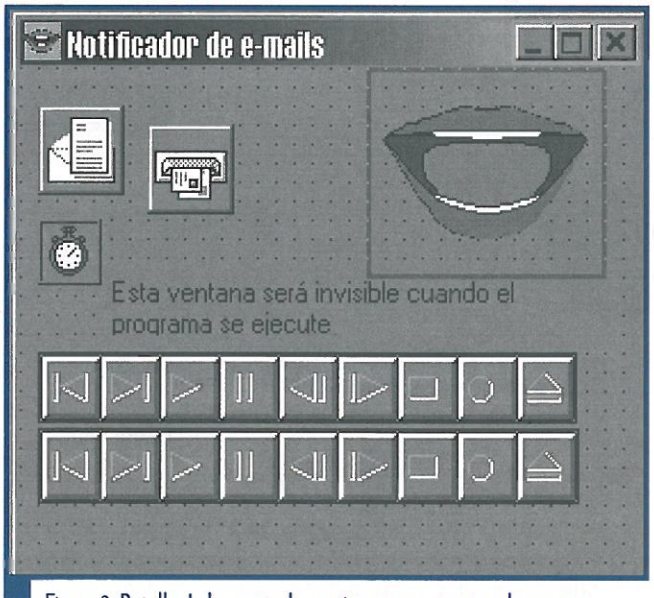


Figura 2. Detalle de los controles activex que componen el programa.

Tabla 1. MAPISession.

Métodos	
SignOff	
SignOn	
Propiedades	
Action	
DownloadMail	Object
Index (Controles ActiveX)	Parent
Index (matriz de controles)	Password
LogonUI	SessionID
Name	Tag
NewSession	UserName

La mayoría de las propiedades del control *MAPIMessages* pueden agruparse en cuatro áreas funcionales:

1. Propiedades de *Libreta de direcciones*.
2. Archivos adjuntos.
3. Mensajes.
4. Destinatarios.

Las propiedades de archivos de datos adjuntos, mensajes y destinatarios están controladas por las propiedades *AttachmentIndex*, *MsgIndex* y *RecipIndex*, respectivamente. Cuando se utiliza el control *MAPIMessages* es necesario hacer un seguimiento de dos *buffers*, el de redacción y el de lectura. El de lectura consta de un conjunto indexado de mensajes traídos de la Bandeja de entrada del usuario. La propiedad *MsgIndex* se utiliza para tener acceso a mensajes individuales dentro de este conjunto, empezando con un valor **0** para el primer mensaje e incrementando en uno cada mensaje hasta el final del conjunto.

El conjunto de mensajes se construye utilizando el método *Fetch*. El conjunto incluye todos los mensajes del tipo *FetchMsgType* y se ordena según se especifique en la propiedad *FetchSorted*. Los mensajes leídos previamente se pueden incluir o dejar fuera del conjunto de mensajes mediante la propiedad *FetchUnreadOnly*. El usuario no puede modificar los mensajes que están disponibles en el *buffer* de lectura, pero sin embargo se pueden copiar al de redacción para modificarlos.

Es posible crear o modificar los mensajes del *buffer* de redacción, el cual se encontrará activo cuando la propiedad *MsgIndex* se establezca a **–1**. Muchas de las acciones de mensajería sólo son válidas dentro del *buffer* de redacción, como enviar mensajes, enviar mensajes con un cuadro de diálogo, guardar mensajes o eliminar destinatarios y adjuntos.

PROPIEDADES Y MÉTODOS DE MAPISESSION

En las siguientes tablas se muestran todos los métodos y propiedades de estos dos controles de mensajería

PROYECTO NOTIFICADOR DE E-MAILS

Para la implementación del ejemplo utilizaremos los controles *ActiveX MAPI* para acceder a los *E-mails*. Cuando el control busque nuevos mensajes, el entorno de Visual Basic se detendrá, y por lo tanto, es mejor ejecutar el programa habiendo sido compilado anteriormente que desde el propio entorno de Visual. Una vez recibidos los mensajes, el programa cortará la comunicación y permitirá que los demás programas sigan ejecutándose normalmente.

Para cambiar la frecuencia con la que se buscan nuevos mensajes, debemos cambiar la constante **POLLTIME** al nuevo valor requerido. *NOTIFYICONDATA* es un tipo de datos necesitado para crear un icono en la bandeja de mensajes recibidos.

```
Private Type NOTIFYICONDATA
    cbSize As Long
    hwnd As Long
    uId As Long
    uFlags As Long
    uCallBackMessage As Long
    hIcon As Long
    szTip As String * 64
End Type
```

Definimos unas constantes para las notificaciones de la bandeja de mensajes:

```
Private Const NIM_ADD = &H0
Private Const NIM_DELETE = &H2
Private Const WM_MOUSEMOVE = &H200
Private Const NIF_MESSAGE = &H1
Private Const NIF_ICON = &H2
Private Const NIF_TIP = &H4
Private Const WM_RBUTTONDOWN =
    &H204
```

Declaramos una función de la *API Windows* para la gestión del icono que notificará nuevos mensajes:

```
Private Declare Function Shell_NotifyIcon Lib
    "shell32" _
  Alias "Shell_NotifyIconA" _
    (ByVal dwMessage As Long, pnid As NOTIF-
    YICONDATA) As Boolean
Dim nid As NOTIFYICONDATA

Public gStarted As Integer
Public gMode As Integer
Public LastRecieved As String
```

Como se ha comentado anteriormente, variando el valor de **POLLTIME** cambiaremos la frecuencia con la que se buscan nuevos mensajes del servidor:

```
Private Const POLLTIME = 10000
```

El procedimiento *Exit_Click* se encargará de eliminar los controles multimedia del programa, así como el icono de la bandeja antes de salir del programa:

```
Private Sub exit_Click(Index As Integer)
MMControl1.Command = "Close"
MMControl2.Command = "Close"
```

Para poder eliminar el icono de la bandeja será necesario utilizar la función **Shell_NotifyIcon**, declarada al inicio del programa.

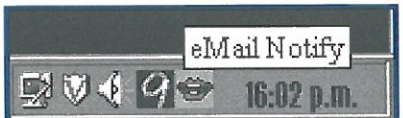


Figura 3. Icono del programa instalado en la barra de tareas de Windows.

Tabla 2. MAPIMessages

Métodos

Compose	ReplyAll
Copy	ResolveName
Delete	Save
Fetch	Send
Forward	Show
Reply	

Propiedades

Action	MsgIndex
AddressCaption	MsgNoteText
AddressEditFieldCount	MsgOrigAddress
AddressLabel	MsgOrigDisplayName
AddressModifiable	MsgRead
AddressResolveUI	MsgReceiptRequested
AttachmentCount	MsgSent
AttachmentIndex	MsgSubject
AttachmentName	MsgType
AttachmentPathName	Name
AttachmentPosition	Object
AttachmentType	Object (Controles ActiveX)
FetchMsgType	Parent
FetchSorted	RecipAddress
FetchUnreadOnly	RecipCount
Index (Controles ActiveX)	RecipDisplayName
Index (matriz de controles)	RecipIndex
MsgConversationID	RecipType
MsgCount	SessionID
MsgDateReceived	Tag (Controles ActiveX)
MsgID	

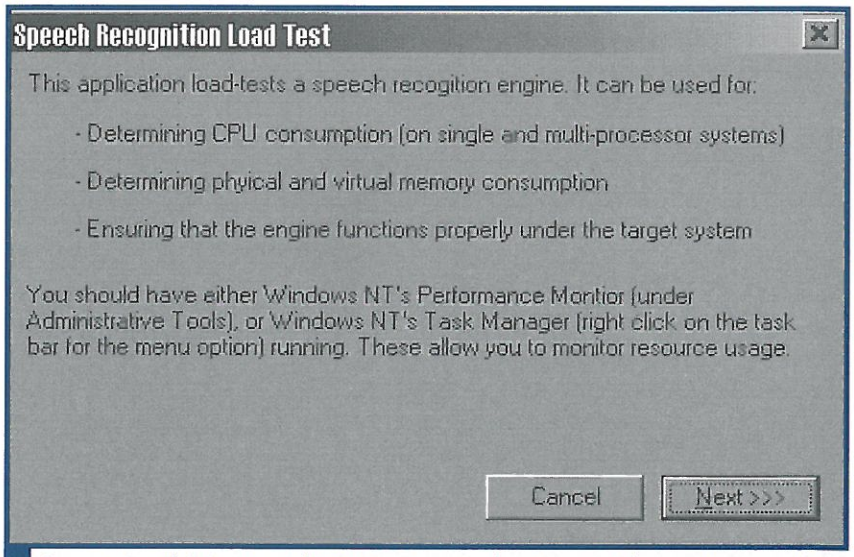

Speech Recognition Load Test
This application load-tests a speech recogition engine. It can be used for:
- Determining CPU consumption (on single and multi-processor systems)
- Determining phyical and virtual memory consumption
- Ensuring that the engine functions properly under the target system
You should have either Windows NT's Performance Montior (under Administratrve Tools), or Windows NT's Task Manager (right click on the task bar for the menu option) running. These allow you to monitor resource usage.
Cancel
Next >>>

Figura 4. La utilidad Loader de la API Speech testea el rendimiento del engine de voz instalado.

```
Shell_NotifyIcon NIM_DELETE, nid
End
End Sub
```

El procedimiento *Form_Load* se ejecuta cuando se carga por primera vez el formulario del programa. Lo primero que debemos hacer es definir el icono de la bandeja de mensajes recibidos. También es necesario especificar el formulario principal, el tipo de icono, etc.

```
Private Sub Form_Load()
nid.cbSize = Len(nid)
nid.hwnd = Form1.hwnd
nid.uId = vbNull
nid.uFlags = NIF_ICON Or NIF_TIP Or
    NIF_MESSAGE
nid.uCallBackMessage = WM_MOUSEMOVE
nid.szTip = "eMail Notify" & vbNullChar
nid.hIcon = Form1.Icon
Shell_NotifyIcon NIM_ADD, nid
```

Debemos inicializar los DOS controles multimedia del formulario para reproducir sonidos que representen nuevos mensajes recibidos. En nuestro caso, utilizaremos los archivos de tipo *WAV* **signon** y **signoff**.

```
MMControl1.notify = False
MMControl1.Wait = True
MMControl1.Shareable = False
MMControl1.DeviceType = "WaveAudio"
MMControl1.filename = App.Path +
    "\signon.wav"
MMControl1.Command = "Open"

MMControl2.notify = False
MMControl2.Wait = True
MMControl2.Shareable = False
MMControl2.DeviceType = "WaveAudio"
MMControl2.filename = App.Path +
    "\signoff.wav"
MMControl2.Command = "Open"
```

No debemos excluir errores cuando el *engine* de texto hablado esté reproduciendo el asunto del mensaje, ya que el usuario puede tener sonidos reproduciéndose. Realmente, deberíamos atrapar esos errores y efectuar la notificación más tarde.

Para ello, utilizaremos el método *SuppressExceptions* del control *ActiveX* de texto hablado y estableceremos el valor **1**. Esto evitará que ocurran excepciones del programa en tiempo de ejecución.

```
DirectSS1.SuppressExceptions = 1
```

Debemos asignar una fecha y hora para el último mensaje recibido. Como ejemplo, podremos una definida, aunque no se corresponda con la realidad.

```
LastRecieved = "10/10/1998 10:00"

gMode = 0
gStarted = 0
Timer1.Interval = POLLTIME
```

Reproducimos un sonido de inicio cada vez que se ejecute el programa (archivo **signon.wav**). Para ello, necesitamos rebobinar el control multimedia **MMControl1** mediante el comando **Prev** y utilizar **Play** para la reproducción:

```
MMControl1.Command = "Prev"
MMControl1.Command = "Play"
End Sub
```

La siguiente función chequea la existencia de nuevos *e-mails*. Sólo debe ser llamada por el **Timer** incrustado en nuestro formulario.

```
Private Sub CheckForNewMail()

Dim SayString As String
```

Debemos tener un poco de precaución. Hay que asegurarse de no comenzar a obtener *mails* si el programa todavía no se ha inicializado. Para ello, utilizaremos la variable **gStarted** que nos indicará la situación actual de la ejecución del programa.

```
If (gStarted <> 1) Then
    Exit Sub
End If
```

La reproducción de sonidos y texto hablado se hará de forma asíncrona

También hay que controlar de algún modo que estamos chequeando nuevos *E-mails*, ya que no queremos repetir este proceso si ya lo estamos haciendo. Esto es necesario porque los sonidos y las palabras habladas por parte del *engine* son reproducidas de forma asíncrona.

```
Rem Obtenemos la nueva lista de mensajes...

MAPIMessages1.FetchSorted = False
MAPIMessages1.Fetch
```

Si hemos encontrado algún mensaje...

```
tCount = 0
again:

If (MAPIMessages1.MsgCount) Then
```

Miramos para saber si es un nuevo mensaje:

```
    MAPIMessages1.MsgIndex = MAPIMes-
      sages1.MsgCount - tCount - 1
    If ((MAPIMessages1.MsgDateReceived)
      >= (LastRecieved)) Then
```

Recordamos la fecha de este nuevo *e-mail* para no leer mensajes anteriores, incluso aunque estén marcados como "no leídos".

```
      LastRecieved = MAPIMessages1.Msg-
      DateReceived
```

Para poder leer los mensajes debemos cambiar de estado el programa (**gMode=2**). Reproduciremos el sonido de aviso **signon.wav**.

```
      gMode = 2
      MMControl1.Command = "Prev"
      MMControl1.Command = "Play"
    Else
```

Si no hay mensajes, salimos del procedimiento:

```
    If (MAPIMessages1.MsgIndex = 0) Then
Else
      tCount = tCount + 1
      GoTo again
    End If
  End If
 Else
 End If
End Sub
```

El siguiente procedimiento indica que el control multimedia ha parado de reproducir el sonido de aviso. Este hecho nos permitirá comenzar a reproducir el asunto del nuevo mensaje mediante el *engine* de texto hablado. Os recuerdo que estamos trabajando de forma asíncrona, y por lo tanto, no podemos reproducir sonidos y texto hablado al mismo tiempo.

```
Private Sub MMControl1_Done(NotifyCode As
    Integer)
```

Si no hay mensajes en la bandeja (**gMode=0**) reproducimos un texto de aviso para notificar al usuario que le avisaremos cuando lleguen.

```
If (gMode = 0) Then
  DirectSS1.Speak "Hola, bienvenido al notifi-
    cador de mensajes. Le avisaré cuando lle-
    guen nuevos mensajes a la bandeja de
    entrada."
```

En caso de que haya nuevos mensajes (**gMode=2**) significa que ya hemos reproducido el sonido de aviso y podemos ahora reproducir el asunto del mensaje. Para ello, utilizaremos el método *Speak* asignándole la cadena de caracteres que contiene el asunto así como la puntilla que queramos agregarle:

```
ElseIf (gMode = 2) Then
SayString = "Tienes un nuevo mensaje de " +
    MAPIMessages1.MsgOrigDisplayName
    + ". Subject line," +
    MAPIMessages1.MsgSubject
```

Reproducimos la cadena de caracteres definida mediante el *engine*:

```
  DirectSS1.Speak SayString
  If (readbody(1).Checked) Then
```

Si hemos leído la parte interesante del mensaje saltamos adelante para no volver a hacerlo, ya que puede resultar pesado.

```
    bodystring = MAPIMessages1.MsgNo-
      teText
    forwardsloc = InStr(bodystring, "——
      Mensaje Original——")

    If (forwardsloc = 0) Then
      forwardsloc = InStr(bodystring, "De:")
    End If

    If (forwardsloc > 0) Then
      bodystring = Left(bodystring, forwards-
      loc - 1)
    End If
    SayString = "Message Body reads: " +
      bodystring
    If (forwardsloc > 0) Then
      SayString = SayString + " Saltando la
      lectura del cuerpo del mensaje."
    End If
    DirectSS1.Speak SayString
```

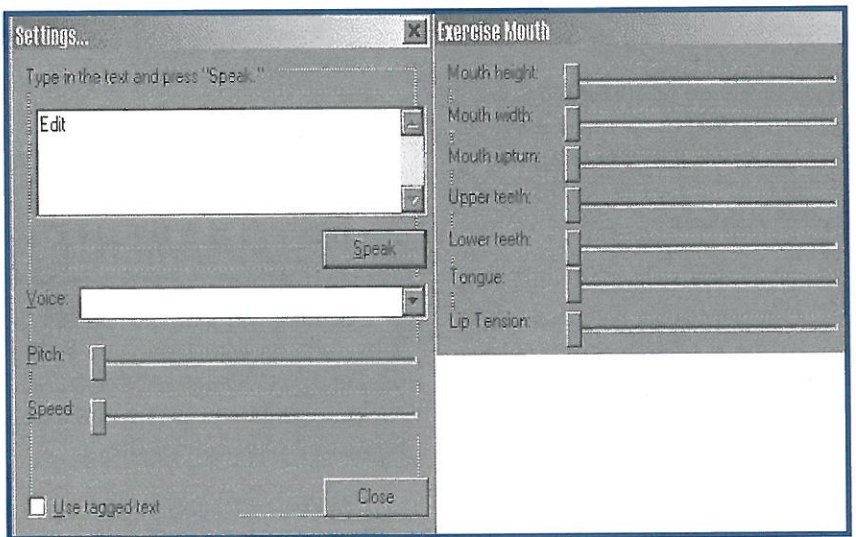


Figura 5. Podemos configurar el engine de voz a nuestro antojo para determinar el tipo de voz que vamos a reproducir.

```
    End If

End If
End Sub
```

El siguiente procedimiento parará automáticamente la lectura de texto por parte del *engine*:

```
Private Sub DirectSS1_AudioStop(ByVal hi As
      Long, ByVal lo As Long)
If (gMode = 0) Then
```

Cuando leemos los nuevos mensajes, también va ser necesario cambiar de estado el programa con gMode

Una vez acabado el proceso de habla por parte del *engine*, reproducimos el sonido **signoff.wav** albergado en el control multimedia **MMControl2** y estamos a la espera de nuevos *E-mails*.

```
  gMode = 1
  MMControl2.Command = "Prev"
  MMControl2.Command = "Play"

  MAPISession1.SignOn
  MAPIMessages1.SessionID = MAPISes-
      sion1.SessionID
```

Activamos **gStarted** para permitir la lectura de los nuevos mensajes que nos lleguen.

```
  gStarted = 1
  Timer1.Interval = 1
  Timer1.Enabled = True
ElseIf (gMode = 2) Then
```

Si estábamos en **gMode=2**, significa que acabamos de leer el asunto del mensaje que hemos recibido. Entonces, podremos reproducir el sonido de finalización (**signoff.wav**):

```
  MMControl2.Command = "Prev"
  MMControl2.Command = "Play"

End If
End Sub
```

Cuando se ejecuta este procedimiento significa que hemos acabado de leer el asunto del mensaje. Por ello, debemos establecer el modo a **3** (variable **gMode**) el cual nos va a permitir la lectura de los nuevos mensajes. que recibamos.

```
Private Sub MMControl2_Done(NotifyCode As
      Integer)

 If (gMode = 2) Then
   gMode = 3
   Timer1.Interval = 1
End If

End Sub


Private Sub readbody_Click(Index As Integer)
  DirectSS1.AudioReset
  readbody(1).Checked = Not readbody(1).Chec-
      ked
End Sub
```

Este procedimiento permite que el ordenador guarde silencio y deje de hablar el contenido del asunto que lleva el mensaje. Para ello, utilizaremos el método *AudioReset* del control *Texto hablado* que hemos insertado en el formulario:

```
Private Sub stopread_Click(Index As Integer)
Rem always stop the current reading so you
      have a way of making the computer shut
      up
DirectSS1.AudioReset
End Sub
```

Con la creación del *Timer* podemos obligar al programa a que busque los nuevos mensajes que nos entren cada cierto intervalo de tiempo, y sin interrumpir las tareas que esté realizando nuestro equipo. Para establecer ese intervalo de tiempo utilizaremos la constante **POLLTIME** que fue definida al comienzo del programa.

```
Private Sub Timer1_Timer()
  Timer1.Interval = POLLTIME
  CheckForNewMail
End Sub
```

La siguiente función se encarga de activar un menú cuando se pulsa el botón del ratón sobre el icono establecido en la barra de tareas de Windows:

```
Private Sub Form_MouseMove _
      (Button As Integer, _
       Shift As Integer, _
       x As Single, _
       y As Single)
```

Para la implementación del ejemplo utilizaremos los controles ActiveX MAPI para acceder a los E-mails

El evento ocurrirá cuando el puntero del ratón esté en el rectángulo que ocupa el ratón y se pulse el botón derecho.

```
    Dim msg As Long
    Dim sFilter As String
    msg = x / Screen.TwipsPerPixelX
    Select Case msg
      Case WM_RBUTTONDOWN
        PopupMenu notify(1)
    End Select
  End Sub
```

PRÓXIMAMENTE

En el siguiente y último artículo de la serie programaremos el control telefónico incluido en el *SDK* de *Speech* para realizar pequeñas utilidades relacionadas con la telefonía.

NOTA: El código que corresponde a toda la información de este artículo se encuentra en el CD-ROM.

www.dadvina.com

Demo disponible en nuestra Web.

Algunos Ratones son muy curiosos...

DadvinaFence

Seguridad en sus Aplicaciones.

Uno de los riesgos que sufren las aplicaciones informáticas es la posibilidad de su uso por personas no autorizadas.
DadvinaFence incorpora tres componentes OCX o VCL:**ApplicationLock**, **FormLock** y **NetMessenger**, proporcionando al programador un sistema de seguridad activo fácil de integrar y totalmente configurable.

- Versiones VCL para Delphi 3.0/2.0, C++ Builder 3 y OCX para Visual Basic 5.0.
- Asignación de permisos y privilegios sobre los objetos existentes en las pantallas de las aplicaciones.
- Autentificación de usuarios mediante el uso de identificación y contraseña.
- Detección de intentos fallidos continuados de entrada al sistema,avisando al instante a los administradores de seguridad de las aplicaciones.
- Centralización de los accesos (login) a las bases de datos de la aplicación.
- Registro de entradas y salidas.
- Sistema modular fácilmente actualizable en tiempo de ejecución y de diseño.

etc...

Sus clientes se lo Agradecerán

Sólo por 31.500* ptas.

Dadvina Systems.

San Toribio 6.
28031 Madrid.
Email: com@dadvina.com

Para más información o realizar pedidos contacte con el telefono 91 380 32 46
o en nuestra Web: www.dadvina.com.

* IVA no incluido.

Construcción de un chat

Javier Sanz Alamillo (jsanz@a01-unix.uc3m.es)

¿Qué usuario de Internet no ha oído hablar de un chat o un IRC? ¿Quién no ha utilizado la atractiva posibilidad de conversar mediante el teclado con otra persona a sabiendas que puede estar en cualquier parte de la Red? Todo esto es posible mediante los programas de chat.

Si hay algo que ha permitido el uso de Internet ha sido poner en contacto a multitud de personas, ya sea gracias al correo electrónico o gracias al chat, un sistema por el cual se pueden comunicar grupos de personas por el envío continuo de mensajes mediante el teclado.

Un Chat es un sistema por el cual se pueden comunicar personas mediante el envío de mensajes por teclado

En este artículo se detallará la forma de crear un *chat*, aunque en principio se implementará la parte correspondiente al cliente, utilizando como herramienta de desarrollo el lenguaje Java. Éste resulta ideal para los desarrollos relacionados con las comunicaciones, tanto por su facilidad de uso como por su potencia en este apartado. Como todo el mundo sabe, una de las características que definen a Java es su clara orientación a la construcción de programas de comunicaciones e Internet (de los cuales todo el mundo conocerá los típicos *applets* de Java).

INTRODUCCIÓN

La construcción de un *chat* es un buen ejemplo de un desarrollo cliente /servidor. La parte cliente representa a un usuario y se encarga de la gestión de todos los aspectos relacionado con el mismo, esto es, el envío de mensajes, la recepción de mensajes de otros usuarios que vayan dirigidos a él, la presentación de la interfaz del usuario, etc.

Por otra parte, el servidor se encarga de todo el trabajo duro, ya que lleva a cabo todo el proceso necesario para poner en contacto a los diferentes clientes, que en el fondo son los distintos usuarios. Además, la parte servidor debe tener la posibilidad de discriminar mensajes, de avisar cuando un cliente ha dejado el *chat*, etc.

Mediante el uso de Java se puede realizar la implementación de las comunicaciones del *chat* utilizando "*streams*", que es un medio "abstracto" por el cual se pueden recibir o enviar los mensajes. Se puede imaginar como unas tuberías, en unas se podrá meter algo que enviar y en otras se puede mirar y comprobar si hay datos, de forma que la gestión de los mensajes se reduce a algo así como "vamos a ver que nos acaba de llegar por aquí ..." o "vamos a enviar algo a través del *stream* ...". De esta forma se aíslan por completo los conceptos de bajo nivel relativos a la comunicación.

La comunicación entre los distintos usuarios del *chat* se realizará mediante *TCP/IP*, lo que simplifica en gran forma todos los problemas de las comunicaciones, ya que no tendremos pérdidas de mensajes, ni alteraciones en los contenidos de los mismos, etc.

Para la implementación de la parte cliente, gracias a Java, podremos elegir

entre desarrollar un *applet* que podremos insertar en nuestra página *web* o una aplicación Java que se podrá iniciar desde nuestro entorno. Siempre queda la posibilidad de utilizar un cliente *telnet* para realizar pruebas de funcionamiento o aprovecharse de las diferentes propiedades que nos puede ofrecer.

La construcción de un Chat es una tarea sencilla en cuanto se conocen los fundamentos para su elaboración

Independientemente del cliente que se va a utilizar, una cualidad que sí es importante se refiere a la portabilidad que ofrece Java para la utilización del *chat*, ya que tanto su parte cliente como servidor pueden ser integradas en cualquier entorno. Así, se puede ejecutar en el clásico Windows, pasando por Linux, por un entorno *Sun*, etc. Para acabar de justificar el uso de Java para la implementación, se debe destacar la facilidad de uso de los *threads*, puesto que nos permitirán realizar el desarrollo de la parte servidor de estilo "*multithreading*. De esta manera se garantiza la creación de un servidor que produce menos carga en caso de que el *chat* soporte cientos de conexiones simultáneas.

La construcción de un chat es un buen ejemplo de un desarrollo cliente /servidor

Si desea obtener más información relacionada con la gestión y uso de *Threads* en Java, o simplemente quiere recordar los conocimientos adquiridos, puede consultar los números 48, 49 y 50 de la revista Sólo Programadores.

CONSTRUCCIÓN DE LA PARTE CLIENTE

El cliente se encarga de todas las tareas necesarias para que el usuario pueda enviar y recibir mensajes, presentándole una interfaz adecuada y encargándose de su gestión. A su vez, el cliente internamente tiene que realizar tareas como ponerse en contacto con el servidor, "conectar", recibir y enviar los mensajes a este servidor, actualizar el contenido de los mensajes que se ven por pantalla, gestionar la desconexión del usuario, etc.

DESCRIPCIÓN DE FUNCIONAMIENTO

Básicamente se puede detallar el funcionamiento del cliente a través del siguiente algoritmo:

```
Presentación de la interfaz de usuario.
Conexión con el servidor.
Mientras haya conexión y el usuario siga conectado
  Comprobación si el usuario quiere enviar algo.
  Si hay algo que enviar entonces
    Envío al servidor.
  Fin Si
  Comprobación si el usuario recibe un mensaje
  Si hay algo que recibir entonces
    Recibir del servidor.
  Fin Si
  Comprobar interfaz si usuario quiere salir.
Fin Mientras
Cerrar conexión con el servidor.
Cerrar interfaz.
Fin.
```

Entrando aún más en detalle, la primera tarea, que resulta ser la presentación de la interfaz de usuario, puede ser resuelta mediante varias opciones:

- Creación de un *applet* para la conexión mediante páginas web.
- Creación de una aplicación clásica Java.
- Utilización de un programa *telnet* que se encargue de todo el trabajo.

La tarea de "conectar con el servidor" implica, ya que se van a utilizar *TCP/IP* y *streams*, la creación de un *socket* a una determinada dirección *IP* y un puerto, y la creación de dos *streams*, uno de entrada y otro de salida. La dirección y puerto del *socket nos* indicarán dónde estará situado nuestro servidor *chat* y donde se conectarán todos los clientes que deseen comunicarse.

Si utilizamos un programa *telnet* clásico, esta opción no presenta problemas, ya que está permitido crear un *socket* con una determinada dirección, pero en caso de utilizar un *applet* nos encontramos con una restricción de seguridad que indica que desde un aquí no se puede crear un *socket* (se lanza una excepción de seguridad). Sin adentrarnos demasiado en detalles, se observará posteriormente que la solución a este problema se basa en indicar a la hora de la creación del *socket* que se trata de uno del estilo conexión *URL*, no un *socket* clásico, lo que evita que se lance la excepción anteriormente citada.

Continuando con las tareas, la siguiente ya es el cliente en sí, puesto que una vez que la conexión se encuentra abierta, sólo hay que conseguir que los mensajes del usuario lleguen al servidor, recibir los mensajes que nos envíen el resto de usuarios y gestionar la interfaz gráfica que tengamos, esto es, indicar que acción hay que realizar cuando se vaya a destruir la conexión, cuando el usuario pulse la tecla **Intro** y por tanto que se envíe el mensaje, o que se actualice el contenido de la ventana que muestra los mensajes recibidos.

Como se habrá observado, nuestro cliente debe tener un mecanismo de comprobación por si hay existen mensajes que enviar o recibir, y a su vez si debe continuar o no, es decir, indicar su finalización. Mediante la programación tradicional *"single-thread"* o único *thread*, el lenguaje debe proveer de una primitiva que permita observar si en un determinado *stream* hay un nuevo dato, o quedan más datos por tratar. A su vez, debe ofrecer una serie de mecanismos para el control de la interfaz. Veamos esta circunstancia mediante el algoritmo:

```
Mientras haya conexión y el usuario
siga conectado
  Si en el stream de datos recibidos hay
  datos
Actualizar la lista de mensajes con estos datos.

  Fin Si
  Si en la interfaz indica que hay un
  datos nuevos
Usar el stream de envío con los datos
  Fin Si
Fin Mientras
```

En este caso, las primitivas de gestión sobre los *streams* son no bloqueantes, es decir, no detienen la secuencia de ejecución hasta que haya algo en el *stream*, sino que se comprueba si hay algo y se continua con la siguiente sentencia. Es un método perfectamente válido y se puede utilizar para implementar el cliente, pero no hace uso de los medios que nos ofrece Java para realizar esta tarea de otra forma más práctica.

Con independencia de la utilización de un *applet* o una aplicación Java normal (*Frame*), se pueden utilizar *threads* para realizar estas tareas de gestión sobre los *streams*, utilizando para ello primitivas de control bloqueantes, esto es, el programa se detiene en la sentencia que gestiona el *stream*, por ejemplo, si tenemos una sentencia del estilo

```
mistream.readLine();
```

La secuencia de ejecución del programa se detendrá en esa línea hasta que se haya recibido un mensaje. Entonces, según el algoritmo anterior, necesitaríamos dos *threads*, uno que se encargara de la lectura del *stream* para la gestión de los mensajes recibidos y otro que se encargara de comprobar si el usuario tiene algo que enviar o se cansó y quiere finalizar.

> La parte servidor debe tener la posibilidad de discriminar mensajes, de avisar cuando un cliente ha dejado el chat, etc.

Para realizar la primera tarea necesitaremos definir un *thread* de forma explícita, pero para la segunda no es necesario, ya que el propio *AWT* se encargará de realizar las comprobaciones adecuadas y sólo se tendrá que comprobar si se cumple un determinado evento, por ejemplo, que el usuario escribió su mensaje y pulsó **Intro**, y en ese caso escribir sobre el *stream* de salida de mensajes.

De forma sencilla, el algoritmo se podría expresar mediante dos *threads* de la siguiente forma:

```
Thread 1, Método propio del thread creado.
    Mientras haya conexión con el servidor
      Leer del stream de datos recibidos
      (bloqueante)
Añadir los datos a la lista de mensaje para el
    usuario
    Fin Mientras

Thread 2, (perteneciente a la interfaz de usua-
    rio) y es el método de gestión de even-
    tos.
    Si el evento es pulso botón de enviar
    datos
      o relleno el área de envío de datos y
      pulso Intro
Escribir en el stream de salida de datos el con-
    tenido del mensaje.
    Fin Si
    Si el evento es destruir la ventana
      Cerrar socket, streams, etc..
    Fin Si
```

DEFINICIÓN DE LA INTERFAZ DE USUARIO

La creación de la interfaz de usuario puede definirse mediante un *applet* para una página *web* o mediante la ya clásica ventana estilo *Frame* de una aplicación Java. De todos modos, principalmente son necesarios estos dos elementos:

- Un campo de texto, donde el usuario tecleará el mensaje que desea enviar.
- Un área de texto donde se irán añadiendo los mensajes recibidos desde el servidor de *chat*.

> Gracias a Java, la implementación de las comunicaciones del Chat es fácil y sencilla, pudiendo utilizar streams

A partir de ese momento, se pueden añadir todos los componentes que se deseen, desde un botón para realizar en envío del mensaje del usuario, un botón para finalizar la conexión con el *chat*, una lista con todos los usuarios conectados, otra con los canales o grupos que están disponibles, etc.

Mediante el *AWT* de Java se pueden construir estos interfaces de una forma muy rápida y sencilla, y mucho mas todavía si se disponen de las herramientasnecesarias como son por ejemplo *RAD* para Java, como *JBuilder*, *VisualAge*, etc., por lo que la creación de una interfaz no supone ningún problema.

DESARROLLO DEL CLIENTE USANDO JAVA

Una vez detalladas las tareas que realizar pasamos a escribir el programa cliente usando Java. Basándonos en la interfaz más simple con los dos componentes elementales que se indicaron, se construye un *applet* que constará de los siguientes métodos:

- *public class cliente extends Applet implements Runnable.*
- *public void run().*
- *public boolean action (Event evt, Object obj).*
- *public void cerrar_socket().*

En el primer método se define el *applet* y se indica que implementará una interfaz de tipo *Runnable*, por lo que se intuye que el *applet* contendrá un *thread* que ejecutará la instrucción dentro del método *run()* como se indica en el segundo método.

El tercer método será el utilizado para gestionar los eventos que ocurran en el *applet*, esto es, el usuario pulsa **Intro**, se añade un mensaje a un área de texto, etc.

El cuarto método es más bien descriptivo e indica que se finaliza la conexión con el servidor, por lo que se cierran los *streams*, el *socket*, etc. La forma en que finaliza el *thread* que se aplica en el *applet* se llevará a cabo mediante el uso de una variable de tipo *boolean* y en caso de finalización total del *applet*, mediante el método *destroy()* del mismo y la invocación al método *stop()* del *thread*.

Aunque pueda parecer que el conjunto de tareas a realizar resulta bastante amplio, como se puede observar en el listado 1 que os presentamos, el resultado final es bastante sencillo, claro y práctico.

Listado 1. Código correspondiente a la parte cliente del chat.

```
import java.awt.*;
import java.applet.*;
import java.io.*;
import java.net.*;

public class cliente extends Applet implements Runnable
{
   final int puerto = 7000;
   TextField campo_entrada;
   TextArea campo_salida;
   DataInputStream   recibido ;
   DataOutputStream  salida ;
   Socket cli_socket ;
   Thread hilo = null;
   String cadena_leida ;
   boolean seguir = true ;

   public void init () {
     campo_entrada = new TextField();
     campo_salida  = new TextArea();
     campo_salida.setEditable(false);
     this.setLayout (new BorderLayout() );
     this.add("South", campo_entrada);
     this.add("Center", campo_salida);
     hilo = new Thread(this);
     hilo.start();
   }
   public void run()
   {
     try  {
       cli_socket = new Socket( getCodeBase().getHost(), puerto );
            salida = new DataOutputStream (
            new BufferedOutputStream                    (
            cli_socket.getOutputStream() ));
            recibido = new DataInputStream          ( new
BufferedInputStream
       ( cli_socket.getInputStream()) );

       while ( seguir ) {
            cadena_leida = recibido.readLine();
            campo_salida.appendText ( cadena_leida +"\r\n");
       }
      }
       catch ( Exception e )  {
          campo_entrada.setEditable(false);
          campo_salida.setText ("No puedo conectar con el servidor de Chat. ");
        }
        finally {
          cerrar_socket();
          hilo = null;
```

Listado 1. (Continuación).

```
        }
      }
      public boolean action (Event evt, Object obj)
      {
        String cadena;
        try {
if (evt.target == campo_entrada &&          campo_entrada.isEditable() ) {
            cadena = (String) obj;
            cadena = cadena.trim();
            salida.writeBytes ( cadena +"\r\n");
            salida.flush();
            campo_entrada.setText("");

            if ( cadena.equals(".s")){
              campo_entrada.setEditable(false);
              seguir = false ;
            }
            return true;
        }
 catch ( IOException e){
campo_entrada.setEditable(false);
            seguir = false ;
        }
       return super.handleEvent ( evt ) ;
      }

      public void cerrar_socket( )
      {
        try {
          if ( cli_socket != null ){
            cli_socket.close();
            recibido.close();
            salida.close();
          }
        catch ( IOException e ) {;}
      }

      public void destroy (){
        if ( hilo.isAlive() ){
          cerrar_socket();
          hilo.stop();
        }
      }
}
```

Según el código, la clase será un *applet* e implementará una interfaz de tipo *Runnable*, utilizado para crear posteriormente un *thread*. Se declaran unas variables, los *streams* de datos, el de entrada y el de salida, y un *string* donde recibir los mensajes. La variable llamada **seguir** se utilizará para finalizar el *thread* hilo de forma que no se invoque a su método *stop()* mientras no sea necesario.

En el método *init*() se construye la interfaz de usuario, se define el gestor de composición, el área de texto, etc. La parte más importante se refiere a la creación y ejecución del *thread* mediante:

```
hilo = new Threat (this);
hilo.start();
```

El método *run()* es el procedimiento que se invocará mediante *hilo.start()* y se encargará de leer los mensajes que se reciban del servidor de *chat,* así como de añadirlos al área de texto para ser visualizados por el usuario. Como se definió anteriormente la lectura es una primitiva bloqueante, es decir, hasta que no se reciba algún mensaje se continúa esperando.

El servidor tiene como tarea la gestión de todos los mensajes de los clientes

El método *action()* es el encargado de gestionar los eventos en la interfaz gráfica, por el cual, cuando el usuario teclea su mensaje y pulsa **Intro** se escribe el mensaje en el *stream* de salida y es enviado al servidor.

También el usuario puede finalizar su conexión, por lo que cuando el evento sea la destrucción del *applet,* se llamará al método *destroy()*, cerrándose el *socket*, los *stream* y deteniéndose el *thread* que lee mensajes. Se puede realizar también una detección del evento *WINDOW_ DESTROY*, pero tiene más sentido cuando el cliente sea la clásica ventana tipo *Frame*.

Como se ha descrito, la creación del cliente es un proceso relativamente sencillo una vez que quedan claramente

definidos los procesos que se deben realizar. Como se ha detallado en los apartados anteriores, todo el proceso se basa en dos tareas muy simples. Gracias a esta definición, se pueden realizar modificaciones en la interfaz sin que estas dos tareas queden afectadas.

CONSTRUCCIÓN DEL SERVIDOR

El servidor tiene como tarea la gestión de todos los mensajes de los clientes. Principalmente esta tarea se basará en recibir un mensaje de un usuario y enviarlo a todos los clientes conectados, de forma que aparentemente parece que lo han recibido directamente.

El funcionamiento de un servidor puede variar en función de que tipo de técnica de desarrollo se vaya a utilizar

El servidor puede realizar más tareas, como un filtraje, es decir, un mensaje puede ser enviado a un único destino, o a un grupo destino, no tiene porque ser recibidos por todos los clientes conectados. También debe realizar tareas de gestión de usuarios, avisar cuando un usuario abandona el Chat, cuando ingresa un nuevo usuario, cuando cambia de nombre o queda temporalmente ausente, etc.

DESCRIPCIÓN DE FUNCIONAMIENTO

El funcionamiento de un servidor puede variar en función de que tipo de técnica de desarrollo se vaya a utilizar. Cada método de desarrollo tiene sus ventajas e inconvenientes, y es el programador en función de sus necesidades, medios e incluso posibilidades el que debe decidir que opción es la más adecuada.

El funcionamiento básico de un servidor se describe de la siguiente forma:

```
Crear un punto de conexión en un puerto.

Seguir <- verdadero
Mientras Seguir = verdadero
  Comprobar si hay nuevas conexiones
  Repetir para todos los clientes conectados
    Comprobar si un usuario ha enviado un mensaje
    Si ha recibido algo entonces
      Filtrar si necesario
      Enviar a todo los usuarios
    Fin Si
Fin Repetir
Actualizar la lista de conexiones si es necesario
Fin Mientras
Cerrar punto de conexión
Fin.
```

Aproximadamente hace lo que se espera, crea un punto de conexión para los clientes, y hasta que se aborte el programa o cualquier otra condición, comprobamos si hay nuevas conexiones de usuarios, por lo que los añadiremos a una lista de conexiones o similar, y se comprueba si alguno o varios de los usuarios han enviado un mensaje, y por cada mensaje recibido enviarlo a todos los clientes conectados. Si en ese momento algún cliente desconecta, se debe actualizar la lista de usuarios conectados, por si por ejemplo, se están realizando búsquedas por usuario o similar.

Si bien la idea principal es bastante sencilla, tiene inicialmente algunos inconvenientes. Por ejemplo, si en algún momento el programa se quedara colgado o se lanzara una excepción, quedaría totalmente bloqueado, por lo que el servidor quedaría inoperativo. Además, al ser un único proceso, éste en caso de cientos de conexiones debe ser lo suficientemente rápido para atender de una forma más o menos razonable todos los mensajes que se reciban, por lo que cabe esperar a que haga un uso intensivo de *cpu*, e incluso encontrándonos que pide al sistema más recursos de los que están disponibles.

Otro problema que puede surgir es determinar cuantas conexiones simultáneas se pueden soportar, ya que a parte de atender las conexiones nuevas, debe gestionar la tarea propia del Chat, que es enviar mensajes a los usuarios. La ventaja clara es que es simple, es un único proceso y si se programa con un lenguaje que ofrezca gestión de excepciones cabe esperar que el servidor no se quede colgado y por tanto este siempre operativo.

Si el lenguaje de programación a utilizar lo permite, se puede aplicar otra idea a la hora de desarrollar el servidor, y el lenguaje Java es ideal para ello. Se basa en la utilización de *threads* por conexión, lo que se denomina la técnica de *thread-per-session*. Se basa en que dado que con el modelo anterior si hay algún problema se puede bloquear todo el servidor, o si es una máquina lenta la que lo ejecuta puede que vaya excesivamente lento, se puede hacer que cada conexión represente a un *thread* del servidor, de forma que este hilo se encarga de recibir los mensajes desde el cliente y enviarlos a todos los demás. El algoritmo elemental se puede describir así:

```
Crear un punto de conexión en un puerto.

Seguir <- verdadero
Mientras Seguir = verdadero
      Si hay una conexión
        Crear hilo que gestione esa conexión,
        Añadiéndolo a la lista de hilos.
      Fin Si
Fin Mientras
Cerrar punto de conexión
Fin.
```

Cada *thread* representa la gestión de una conexión cliente, por lo que se puede representar su funcionamiento de la siguiente manera:

Listado 2. Listado de la clase.

```
import java.util.*;
import java.io.*;
import java.net.*;

public class servidor {
   static Vector v;
   public final static String CADENA_SALIR = new String(".s");
   public final static String CADENA_LISTA_USU = new String(".u");
   public static synchronized void manejaMensaje
            ( String nombre_usu, int id_hilo, String cad )
   {
      try {
            if ( cad.equals(CADENA_SALIR )){
                  salir_usu( id_hilo );
            }
            else {
                  if ( cad.equals(CADENA_LISTA_USU )){
                        lista_usuario (id_hilo);
                  }
                  else {
                        mandaMensajeTodos(nombre_usu+"> "+cad);
                  }
            }
      }
      catch ( StringIndexOutOfBoundsException e ) {
   mandaMensajeUno( "Por Dios, una silaba al menos ..." ,id_hilo);
      }
   }
   public static synchronized void mandaMensajeTodos ( String cad )
   {
       for ( Enumeration e = v.elements() ; e.hasMoreElements(); ){
            objetoServidor OB=(objetoServidor)e.nextElement();
            OB.enviar_cad (cad);
      }
   }
   public static synchronized void mandaMensajeTodosExcepto (     String cad, int
id_hilo )
   {
      for ( Enumeration e = v.elements() ; e.hasMoreElements();){
            objetoServidor OB=(objetoServidor)e.nextElement();
            if ( OB.id_hilo != id_hilo ) {
                  OB.enviar_cad (cad);
            }
      }
   }
public static synchronized void mandaMensajeUno( String cad, int id_hilo )
   {
   for ( Enumeration e = v.elements() ; e.hasMoreElements() ;){
objeto Servidor OB = (objeto Servidor)e.nextElement ()
if (OB.id_hilo==id_hilo){
```

```
Mientras haya conexión con el cliente
      Esperar un mensaje del cliente (bloqueante)
        Filtrar el mensaje
        Enviar el mensaje al resto de clientes,
        Recorriendo la lista de hilos activos.
Fin Mientras
Finalizar thread, eliminándolo de la lista
```

Como se aprecia en este último modelo, cada *thread* representa una conexión, por lo que en caso de ocurrir un problema en alguno de ellos solamente se perdería esa conexión, en ningún momento el resto de clientes quedarían afectados por la misma. Debido a que en vez de un proceso que hace muchas cosas, tenemos varios procesos que hacen una específica, cabe esperar que su ejecución sea más eficiente.

La clase servidor contiene la lista con todos los threads que gestionan las conexiones y son objetos de tipo objetoServidor

Además de ventajas, también tiene inconvenientes. El más importante es el que acompaña a la programación con *threads*, y son los problemas de concurrencia, interbloqueos, etc., que hacen que la búsqueda de errores sea difícil y a veces casi imposible. Otro problema es el tiempo que se gasta en la creación y destrucción de los hilos. Aunque es un tiempo bastante menor al consumido para crear un proceso tradicional, aún así es considerable, y en el caso de la creación múltiples *threads* puede llegar a ser una fuente de consumo de recursos importante.

Una mejora a la técnica anterior, como solución a los tiempos de creación y destrucción de los *threads,* es generar tantos *threads* como conexiones se quieran tener y utilizar cada

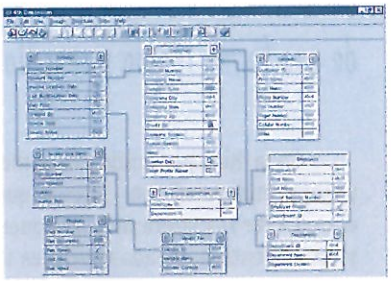
Si le gustaron otras bases de datos, 4th Dimension v6 *le sorprenderá*

4th Dimension es el entorno de bases de datos integrado más completo del mercado. Años de experiencia en el diseño de bases de datos han permitido satisfacer las siguientes necesidades:

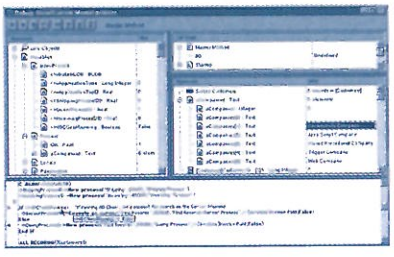
- Automatizar cualquier proceso.
- Diseñar modelos gráficos eficientes y orientados a objeto.
- Suministrar la más amplia gama de herramientas para facilitar el diseño y mantenimiento de bases de datos.
- Permitir una fácil reutilización de código.
- Permitir escalar desarrollos Cliente/Servidor, web y multiplataforma de forma transparente.

Estas son algunas de las **características** de 4th Dimension:

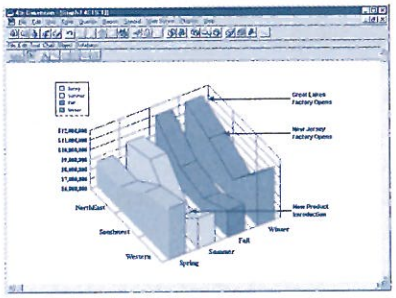
- **Motor de base de datos multiproceso, triggers, procedimientos almacenados**, presentación visual de tablas mediante diagrama de entidad /relación (ERD)



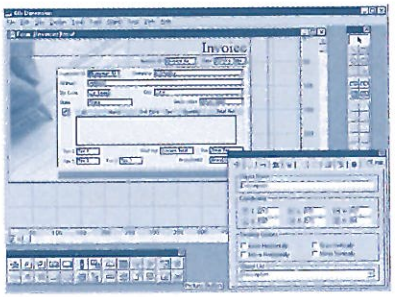
- **Debugger visual** multiproceso de última generación. **Intérprete** incluido para testeo y debugging.



- **Wizards avanzados** para diseño de formularios, queries, informes, importación, exportación, etiquetas, etc. Editor de gráficas programable 2D y 3D, soporta gráficas de barras, tartas, áreas, líneas, etc.



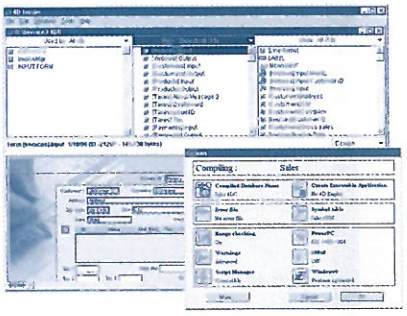
- **Editor de formularios basado en objetos.** Gestión de hojas de estilo. Redimensionamiento automático. **Docenas de objetos de interface** incluyendo check boxes, radio buttons, push buttons, áreas de scroll, picture button, botones invisibles, tab control, listas jerárquicas, termóme-tros, reglas, combo box, listas menú desplegables, menús pop-up jerárquicos, menús picture, radio pictures, listas desplegables, menús pop-up, subforms, group boxes, etc.



- Arquitectura de distribución de aplicaciones integrada y **soporte para proyectos con múltiples desarrolladores** con check-in/check-out automatizado.
- Arquitectura extensible mediante **plug-ins** y DLLs.

- **Un sólo código fuente** desplegable sin modificaciones de monousuario a 4D Server, la versión Cliente/Servidor de 4th Dimension.
- Soporte de **ODBC** (cliente y servidor) y conexiones nativas **SQL** a ORACLE, Sybase y MS SQL Server.
- **Servidor Web integrado** para acceso automático a formularios y gestión Cliente/Servidor mediante navegadores web.

- **Compilador a código máquina real** optimizado para diferentes procesadores. **Gestor gráfico de referencias cruzadas** y librerías de objetos.



VERSIÓN DEMO GRATUITA

4th Dimension es el núcleo de la gama de productos 4D. Visite http://www.ambitconsulting.com o llame al 93-209 41 11 para más información o para recibir una versión demo gratuita. Mencione este anuncio y reciba 4th Dimension v6 por el precio promocional de 34.900 Ptas*. (precio habitual: 79.000 Ptas.).

AMBIT Consulting
Balmes, 297, 4º, 2ºb. 08006 Barcelona
t. | (34) 93 209 41 11 f. | (34) 93 240 18 88
e-mail: ambit@ambitconsulting.com
http://www.ambitconsulting.com

BelSoft
Bellsoft, S.L. C/Salva, 4-2º-2ª 46002 - Valencia
t. | (34) 96 352 71 30 f. | (34) 96 352 71 30
AppleLink SPA0351 e-mail: bellsoft@combios.es
http://www.bellsoft.es

[* Actualización competitiva. Oferta válida hasta el final de existencias]

Listado 2. (Continuación).

```
                    OB.enviar_cad (cad);
                }
            }
        }
        public static synchronized void lista_usuario ( int id_hilo){
String mensaje ;
            mensaje = new StringBuffer().
                append("Usuarios Conectados "+"\n\r").
                append("---------------------- ").toString();
            mandaMensajeUno ( mensaje, id_hilo);
            for ( Enumeration e = v.elements(); e.hasMoreElements(); ){
                objetoServidor OB = (objetoServidor) e.nextElement();
                mandaMensajeUno ( OB.nombre_usu + " <desde> "+ OB.dir_usu ,
                id_hilo);
            }
        }
        public static synchronized void salir_usu ( int id_hilo_salir ){
            for ( Enumeration e = v.elements() ; e.hasMoreElements();){
                objetoServidor OB = (objetoServidor)e.nextElement();
                if ( OB.id_hilo == id_hilo_salir )    {
                    borrar_usu (id_hilo_salir , OB.nombre_usu );
                    OB.enviar_cad (" A la que te cae majo \n\r ");
                    OB.cerrarSocket();
                    OB.cerrarThread();
                }
            }
        }
        public static synchronized void borrar_usu ( int id_hilo, String nombre_usu ){
        for ( Enumeration e = v.elements(); e.hasMoreElements();){
                objetoServidor OB = (objetoServidor)e.nextElement();
                if(OB.id_hilo == id_hilo){
                    v.removeElement ( OB );
                    if( nombre_usu != null )
                        mandaMensajeTodosExcepto ( nombre_usu+" se
marcho...",id_hilo);
                }
            }
        }
        public static void main( String args[] )  {
            int serv_puerto ;
            int cont_conex;
            try {
                v = new Vector();
                cont_conex = 0;
                serv_puerto =Integer.parseInt(args[0]) ;
                ServerSocket sock_servidor = new ServerSocket ( serv_puerto) ;
                while ( true ) {
                    Socket cli_sock = sock_servidor.accept();
                    v.addElement ( new objetoServidor
                        (cont_conex, cli_sock, cli_sock.getInetAddress().toString()))
```

thread para cada nueva conexión, evitando así su destrucción. Aunque con esta técnica denominada *Thread Pool* se ahorran los tiempos relacionados con los hilos, aún se mantienen los problemas de concurrencia con el añadido de decidir cuantos hilos deben crearse, y si se establece un método de control del máximo de conexiones de forma que se de un aviso en cuanto no se soporten más clientes. Se puede describir este método de la siguiente forma:

```
Crear un punto de conexión en un puerto.
Seguir <- verdadero
Para el numero de conexiones simultaneas
        Crear hilo correspondiente
        Almacenarlos en una lista, tabla, etc...
Fin Mientras
Esperar a la finalización de todos los hilos
Cerrar punto de conexión
Fin.
```

Mediante el AWT de Java se pueden construir estos interfaces de una forma rápida y sencilla

Cada hilo generado es como un proceso independiente, puesto que se encarga de la espera de una conexión y de su gestión, y cada *thread* es utilizado para las siguientes conexiones cuando la actual finalice. Por tanto, cada hilo ejecutará la siguiente porción de código:

```
Esperar una conexión
Mientras haya conexión con el cliente
        Esperar un mensaje del cliente (bloqueante)
        Filtrar el mensaje
        Enviar el mensaje al resto de clientes
        Recorriendo la lista de hilos activos.
Fin Mientras
```

Como se aprecia, es el hilo el que realiza la espera de un nuevo cliente, por lo que el problema que se tenía con el primer modelo sobre cuántas

Listado 2. (Continuación)

```
                    mandaMensajeUno ( "Nombre : " ,cont_conex );
                    cont_conex++;
                }
            }
            catch ( IOException e )     {
                e.printStackTrace();
            }
            catch ( IndexOutOfBoundsException e ) {
                e.printStackTrace();
        }
        }
    }
```

conexiones simultáneas se pueden aceptar aquí está perfectamente definido, ya que todos los hilos están esperando una conexión, y se asignará según el orden de ejecución que corresponda en un momento dado. No aparecen en ningún momento referencias a la destrucción del *thread*, ya que el mismo es reutilizado por diferentes clientes en distintos periodos de conexión.

La clase objetoServidor es la encargada de la gestión individual de cada conexión

El principal problema que plantea este modelo, a parte de la concurrencia, es definir claramente el número de *threads* a generar inicialmente, y qué debe ocurrir cuando todas las conexiones estén ocupadas, si se genera un grupo auxiliar de conexiones, si se encolan en una espera las conexiones, por lo que los *threads* deben primero mirar en la cola antes de esperar a una conexión, etc. Como se puede observar, también se pueden generar numerosas alternativas a este modelo.

Una modificación que se puede aplicar a los métodos que utilizan *threads* es similar a la que se aplicó con el *applet* al utilizar diferentes *threads* en diferentes tareas. En este caso, en vez de utilizar un único *thread* para la gestión de la conexión, se pueden crear dos *thread* para realizar la misma tarea. Un *thread* se encargaría de la recepción de los mensajes del cliente y otro de enviar los datos al mismo, de forma que mientras un mensaje se está enviando a los clientes estos a su vez pueden estar recibiendo un mensaje de otro cualquiera.

Esta modificación da una idea de ganancia de tiempo de ejecución considerable, pero hay que reconocer que su implementación hace que si ya con tantos *thread* por conexión la corrección de errores es complicada y los problemas que pueden ocurrir de interbloqueos y concurrencia pueden ser importantes, con esta modificación puede ser ya algo serio y en algunos casos casi imposible de manejar.

IMPLEMENTACIÓN DEL SERVIDOR

Una vez que se han detallado una serie de ideas para la implementación del servidor vamos a pasar a la práctica utilizando la técnica de *thread-per-session*, y guardando los *threads* en una lista. Gracias al uso de Java, la creación de los hilos es una tarea casi trivial y la gestión de la lista es un proceso elemental gracias a los objetos de tipo Vector() que nos ofrecen todas las funcionalidades que son necesarias.

El servidor consta de dos clases:

- servidor
- objetoServidor

La clase *servidor* contiene la lista con todos los *threads* que gestionan las conexiones y son objetos de tipo *objetoServidor*. Esta clase se encarga de todas las tareas de envío de mensajes a los clientes, gestión de los tipos de mensajes, etc.

El método *main*() se encarga de aceptar las conexiones y a partir de ese momento lanzar un objeto de tipo *objetoServidor*, que es añadido a la lista de clientes conectados. Mediante el método *manejaMensaje* se determina que acción se debe realizar según el mensaje, ya que a parte de poder enviar un mensaje al resto de usuarios se puede generar una lista con el nombre de los usuarios conectados, su origen, etc., como el método *lista_usuario*.

Las estructuras necesarias para el servidor se basan en crear nuevos objetos y una estructura tipo Vector()

El resto de los métodos se encargan de realizar los envíos a los clientes recorriendo la lista de conexiones, como *mandaMensajeTodos*(), y recorriendo la lista invoca al método *enviar_cad*() de cada *objetoServidor* y por tanto, ese mensaje es enviado a cada cliente. También se han creado otros dos métodos , *mandaMensajeTodosExcepto*() y *mandaMensajeUno*() que realizan unas tareas más específicas que se entienden observando detenidamente el código.

Listado 3. Código correspondiente a la parte del servidor.

```
class objetoServidor implements Runnable {
   private Thread hilo = null;
   private Socket sock = null;
   private DataOutputStream salida = null;
   private DataInputStream  recibido = null;
   private String buffer;
   public  String dir_usu ;
   public  String nombre_usu;
   public  int id_hilo;

objetoServidor ( int indice, Socket sock, String dir_conex){
        this.sock = sock;
        this.id_hilo = indice;
        dir_usu = new String ( dir_conex );
        try    {
        salida = new DataOutputStream ( sock.getOutputStream() );
        recibido = new DataInputStream ( sock.getInputStream() );
        hilo = new Thread (this);
        hilo.start();
        }
        catch ( Exception e )          {
              e.printStackTrace();
        }
   }
   public void run(){
        try {
              nombre_usu = recibido.readLine();
  servidor.mandaMensajeTodos    (" Conectado ... " + nombre_usu );
         while(true) {
              buffer = recibido.readLine() ;
              buffer = tratar_buffer ( buffer );
              servidor.manejaMensaje(nombre_usu, id_hilo, buffer );
              }
        }
        catch ( Exception e )          {
              servidor.borrar_usu(id_hilo,nombre_usu);
              cerrarSocket();
              cerrarThread();
        }
```

El método *salir_usu()* se encarga de la gestión de la conexión en su fase de eliminación, ya que hay que eliminar la conexión de la lista de clientes, detener el *thread* de *objetoServidor*, etc. A continuación tenemos el listado de la clase.

La clase *objetoServidor* es la encargada de la gestión individual de cada conexión, ya que lanza el *thread* en sí, que controla todo lo relativo a los mensajes del cliente.

El método *run*() relativo al *thread* generado básicamente se encarga de permitir la conexión con el cliente tras la aceptación por el método *main*() de la clase servidor. Espera un mensaje de forma bloqueante y una vez recibido invoca el método de gestión de la clase servidor que realizará la tarea correspondiente. El mensaje puede ser previamente adaptado a nuestras necesidades, ya sea comprobando su tamaño, eliminando caracteres no visualizables, etc.

Si el lenguaje de programación a utilizar lo permite, se puede aplicar otra idea a la hora de desarrollar el servidor

Otro método importante es *enviar_cad*() que envía un mensaje al cliente que está adjudicado al thread. Este método será invocado cada vez que un mensaje tenga que ser enviado al usuario.

El resto de procedimientos, como *cerrarSocket*() y *cerrarThread*() se encargarán de la finalización de la conexión con el cliente y la eliminación en sí del *thread* generado, por lo que el objeto de tipo *objetoServidor* será destruido y con él la conexión del usuario al Chat.

Gracias al uso de Java, el desarrollo con hilos suele ser una de las mejores opciones, no sólo por su potencia sino por su simplicidad de uso

Como se ha podido comprobar a lo largo del listado, la creación del servidor es un proceso que más que dificultad requiere de un cierto tiempo y constancia. Se pueden aplicar para el desarrollo del servidor diferentes técnicas así como también añadir nuevas funcionalidades.

NUEVAS FUNCIONALIDADES

Gracias a la estructura del servidor, se pueden añadir nuevas funcionalidades sin más que añadir el método que realice la tarea y añadir al método *manejaMensaje*() la correspondiente entrada de identificación. Si por ejemplo deseamos que un usuario pueda recibir una ayuda si teclea ".h" simplemente se tendría que añadir:

```
public static synchronized void enviar_ayuda(int
        id_hilo){
        String mensaje ;
        mensaje = new StringBuffer().
        append("AYUDA          "+"\n\r").
        append("Comando :      "+"\n\r").
        append("Salir -> .s  <- que original
        "+"\n\r").
        append("Lista conectados -> .u <- te
        suena "+"\n\r").
        append("Oculto -> .o usuario mensaje
        "+"\n\r").
        append("Ayuda -> .h "+"\n\r").toString();
        mandaMensajeUno( mensaje, id_hilo);
}
```

y en el método *manejaMensaje*() se añadiría:

```
public final static String CADENA_AYUDA =
        new String(".h");
if ( cad.equals (CADENA_AYUDA )) {
        enviar_ayuda(id_hilo);
}
```

Añadir otras características pasaría por repetir este proceso, así de fácil.

CONCLUSIONES

A lo largo del artículo se ha mostrado como funciona un Chat, tanto en su parte cliente, como en su parte servidor. Se han mostrado diferentes técnicas para el desarrollo del servidor, como *thread-per-session*, único proceso, etc. Las estructuras necesarias para el servidor se basan en crear nuevos objetos y una estructura tipo Vector() que gestiona las altas, bajas, etc. de las conexiones cliente. La parte cliente también puede ser implementada de varias formas, pero gracias al uso de Java, el desarrollo con hilos suele ser una de las mejores opciones, no sólo por su potencia sino al mismo tiempo por su simplicidad de uso.

BIBLIOGRAFÍA

- "Thinking in Java". Bruce Eckel. Prentice Hall.
- "Exploring Java". Patrick Niemeyer, Joshua Peck. O'Reilly & Associates Inc.
- "Java in a nutshell". David Flanagan. O'Reilly& Associates Inc.

Listado 3. (Continuación).

```
}
public String  tratar_buffer ( String buffer )     {
    if ( buffer.length() > 79 )     {
        buffer = buffer.substring (0,79 );
    }
    buffer = buffer.trim();
    return buffer ;
}
public void enviar_cad ( String cad_usu ) {
    try{
        salida.writeBytes ( "\r\n"+cad_usu+"\r\n"  );
        salida.flush();
    }
    catch ( Exception e )          {
        servidor.borrar_usu ( id_hilo, nombre_usu );
        cerrarSocket();
        cerrarThread();
    }
}
public void cerrarSocket() {
    try    {
        if ( sock != null ) {
            sock.close();
            salida.close();
            recibido.close();
        }
    }
    catch ( Exception e )          {
        e.printStackTrace();
    }
}
public void cerrarThread() {
    try    {
        if ( hilo != null && hilo.isAlive() )
            hilo.stop();
    }
    catch ( Exception e )          {                    e.printStackTrace();
    }
}
}
```

Planificación de procesos

Juan Manuel Menéndez (jfrias@ibm.net)

Linux posee un eficaz sistema de planificación de procesos basado en tres colas, dos de ellas para procesos de tiempo real y la tercera para otros tipos de procesos. En este artículo se verá de una forma práctica cómo se puede trabajar con las prioridades, pero antes se explicarán unos conceptos básicos.

Generalmente se suele definir un proceso como un programa que se está ejecutando, aunque de forma más correcta se puede considerar que un proceso está formado por el programa que se ejecuta, sus datos (en memoria dinámica o estática), su pila, el valor de los registros generales de la *CPU* y toda la información de control relativa al proceso que está en posesión del sistema operativo. De estos elementos, el programa y los datos, se encuentran contenidos en un rango de direcciones de memoria virtual que va desde 0 hasta el máximo valor que soporte el sistema operativo donde se ejecuta. La información que tiene el sistema operativo sobre el proceso se denomina *atributos del proceso*.

Entre los atributos de un proceso se encuentran:

- Identificador (*PID*)
- Identificador del proceso padre (*PPID*)
- Valor de los registros
- Identidades del usuario y grupos.
- Prioridad del proceso con respecto a otros.
- Recursos consumidos por el proceso.
- Ficheros abiertos, mecanismos *IPC* y cualquier otro tipo de recurso que suministre el sistema operativo.
- Estado del proceso.

Pasemos a explicar de forma breve cada uno de estos conceptos antes de entrar en el tema principal del artículo, orientado a la planificación de procesos.

El atributo que define unívocamente a cada proceso es el identificador de proceso (o *process id*), más conocido por su acrónimo sajón *PID*. Se trata de un número único que el sistema asigna a cada proceso de tal forma que aunque los *PID* se puedan repetir en el sistema, nunca pueden hacerlo en el mismo momento. Este *PID* es el único valor que tendremos para dar órdenes a los procesos, ya sea mediante funciones y llamadas al sistema desde un programa, o desde la línea de comandos de la *shell*.

Una característica de los sistemas Unix (y por tanto de Linux) consiste en que todos los procesos son arrancados por otros procesos, por lo que se establece una jerarquía que se puede observar mediante el comando *ps* con el parámetro *f*, tal y como se muestra a continuación:

```
ps f[otros parámetros]
```

Las características que tiene un proceso dependen en parte de las propias características que tenga el sistema operativo, así por ejemplo tanto Windows 95 como OS/2 son sistemas multiproceso y monousuario, mientras que Linux se caracteriza por ser un sistema multiproceso y multiusuario. Los sistemas multiusuario permiten asignar a cada recurso del sistema un comportamiento diferenciado para cada usuario, de tal forma que parece lógico suponer que este comportamiento sea extendido a los procesos arrancados por los usuarios. Así pues los procesos tienen dos usuarios: el real y el efectivo. El usuario real es aquél que ha lanzado el proceso,

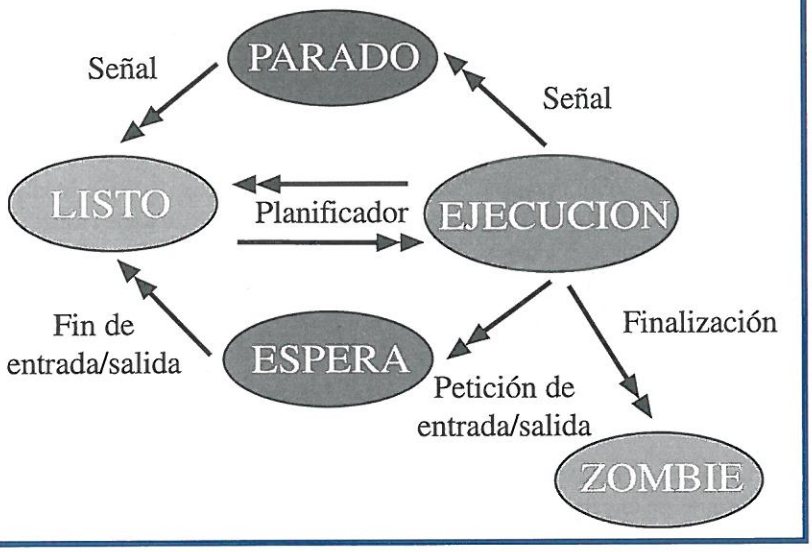


Figura 1. Diagrama de estado de un proceso y descripción de eventos .

mientras que el efectivo es el que utiliza el sistema para controlar el acceso a los recursos. La forma de actuar con respecto a los grupos es análoga a la comentada para los usuarios.

El estado de un proceso en un momento determinado indica qué actividad está realizando en ese momento. Los posibles estados en que puede estar un proceso son:

- *Ejecutándose*: Los registros generales de la *CPU* están cargados con los valores correspondientes a este proceso y la *CPU* está ejecutando una instrucción del mismo.
- *Preparado*: El proceso puede ser ejecutado, pero existe otro proceso en ejecución en ese momento.
- *En espera*: El proceso está a la espera de un recurso, por ejemplo en espera de una entrada/salida.
- *Parado*: El propio proceso ha solicitado su parada.
- *Zombi*: El proceso ha finalizado pero el sistema operativo sigue manteniendo toda la información relativa al mismo.

Obviamente en un sistema con un único procesador solamente puede haber un proceso en ejecución, mientras que los demás se encuentran en estado *Preparado* formando una cola de espera. Más adelante se tratará el tema de la organización de los programas colocados en la cola de espera.

LOS PROCESOS DESDE LA SHELL

Los comandos más utilizados para la gestión de los procesos son *ps* y *kill*. Mediante el primero de ellos se puede obtener una visión general del estado de los procesos arrancados en el sistema, aunque presenta una gran variedad de salidas dependiendo de los parámetros que se le suministren, siendo los más habituales:

- *l*: formato largo, muestra *PID*, *PPID*, tiempo de *CPU* consumido, etc.
- *f*: formato en árbol jerárquico.
- *x*: muestra los procesos que no tienen terminal de control.

Con el comando *kill* es posible mandar una señal al programa. Una señal es una técnica mediante la cual es posible alterar la secuencia normal del programa desde el exterior. En Linux existen **32** señales cada una de ellas con un uso muy específico, pero aunque es posible mandar cualquier señal desde la *shell* con el comando *kill*, lo más usual es utilizar este comando para finalizar un proceso mediante las señales **15** (*SIGTERM*) o **9** (*SIGKILL*) de la forma:

```
kill 15 PID_proceso
```

> Las propiedades que tiene un proceso dependen en parte de las propias características que tenga el sistema operativo

La forma correcta de finalizar un proceso mediante *kill* consiste en utilizar la señal *15*, pues utilizando esta señal se da la posibilidad al proceso de realizar un cierre ordenado del mismo, mientras que si se manda la señal *9* las posibilidades del proceso para realizar un cierre ordenado son bastante pocas. Si después de enviar la señal *15*, y tras un tiempo prudencial (este tiempo dependerá de cada proceso y sólo la experiencia puede servir de guía), el proceso en cuestión no ha finalizado, entonces se deberá recurrir a la señal *9*.

LOS PROCESOS DESDE LA PROGRAMACIÓN

Desde un programa y usando diversas llamadas al sistema se pueden realizar las siguientes tareas con procesos:

- Arrancar un proceso hijo.

- Esperar la finalización de uno o varios procesos hijos.
- Obtener y modificar los atributos de los procesos.
- Obtener y modificar las prioridades de planificación.

De todo este abanico de posibilidades se va a explicar únicamente el proceso de creación de un proceso hijo mediante la llamada al sistema *fork*, así como las estructuras que definen a los procesos dentro del sistema operativo con el fin de sentar las bases necesarias para comprender el funcionamiento del planificador de procesos del sistema operativo. De esta manera se podrán seguir los programas de la práctica con el máximo aprovechamiento.

Para crear un proceso se emplea la llamada al sistema *fork*. Cuando un proceso ejecuta esta llamada, el sistema operativo duplica el proceso que la ha ejecutado, tanto la parte de código, memoria estática, memoria dinámica y la pila, de tal forma que una finalizada la llamada ambos procesos están listos para tomar el control de la *CPU*.

Según lo expuesto tenemos dos procesos que pueden coger el control de la *CPU*, y ambos son idénticos, con lo cual cuando se ejecuten ambos procesos continuarán haciendo lo mismo. Si estos procesos estuvieran leyendo un fichero ambos leerían el mismo registro, realizarían el mismo proceso sobre los registros, y escribirían el mismo resultado sobre el mismo fichero, no hay que olvidar que al duplicarse el proceso se duplican los descriptores de ficheros y por lo tanto apuntan a los mismo ficheros. No parece tener mucho sentido, ya que lo más lógico es que si se crea un proceso hijo éste sirva para realizar una tarea distinta.

La forma de controlar esta situación después de una llamada al sistema *fork* consiste en determinar qué proceso de los dos es el padre y cuál es el hijo. Esta distinción se consigue basándose en el código de retorno de la llamada al sistema utilizada, que puede ser uno de los tres siguientes:

- *Igual 0*: Este código lo recibe el proceso hijo.
- *Mayor que 0*: Este código lo recibe el proceso que ha ejecutado la llamada al sistema, es decir el proceso padre. Y su valor es el número de proceso que identifica al proceso hijo obtenido como resultado de la llamada.
- *Menor que 0*: La llamada al sistema no ha podido crear un proceso. Este código lo obtiene obviamente el proceso que ejecuta la llamada.

Basándonos es estos códigos la codificación que se debe realizar debe tener un aspecto similar al siguiente:

```
rc = fork()
if (rc < 0)
{
      /* Llamada incorrecta */
      exit;
}
if (rc > 0)
{
      /* Llamada correcta bloque de programa
      que ejecuta
      el proceso padre  rc contiene el número
      de proceso
      de hijo
      */
{
else
{
      /* Parte del programa que ejecuta el proceso hijo */
}
```

Otro punto que puede ser tan importante como la creación de procesos hijos es el control de la finalización de los mismos. Un proceso puede esperar la finalización de un proceso hijo de dos formas: síncrona o asíncrona.

Con la forma síncrona el proceso está esperando la finalización de un proceso hijo determinado o un proceso hijo cualquiera mediante la llamadas al sistema: *wait* y *waitpid*. La primera alternativa espera la finalización de un proceso hijo cualquiera, mientras que la segunda resulta más interesante, ya que permite esperar a la finalización de un proceso hijo determinado según los valores que se le suministre a esta llamada. El prototipo de la función es el siguiente:

```
waitpid(pid_t pid, int statsup, int options)
```

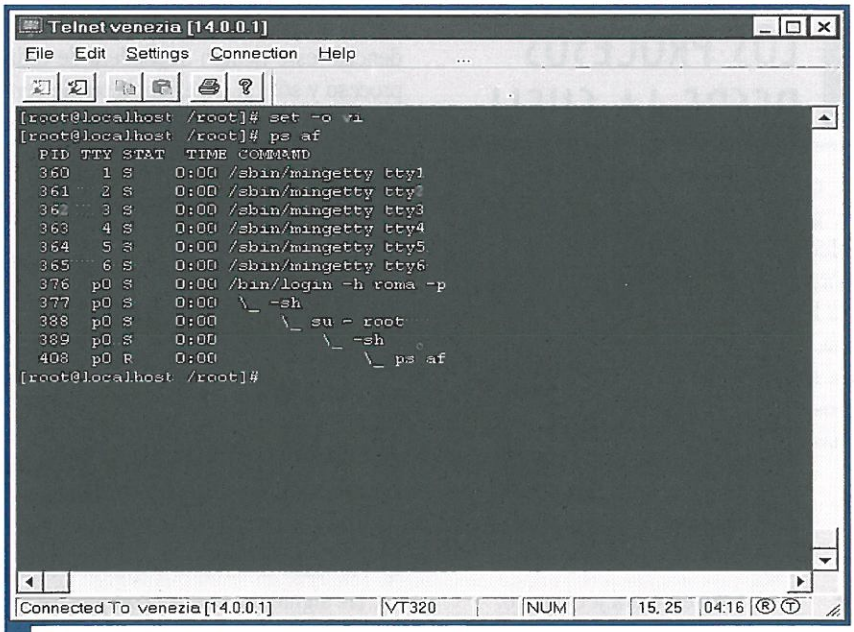

Telnet venezia [14.0.0.1]
File Edit Settings Connection Help
[root@localhost /root]# set -o vi
[root@localhost /root]# ps af
PID TTY STAT TIME COMMAND
360 1 S 0:00 /sbin/mingetty tty1
361 2 S 0:00 /sbin/mingetty tty2
362 3 S 0:00 /sbin/mingetty tty3
363 4 S 0:00 /sbin/mingetty tty4
364 5 S 0:00 /sbin/mingetty tty5
365 6 S 0:00 /sbin/mingetty tty6
376 p0 S 0:00 /bin/login -h roma -p
377 p0 S 0:00 _ -sh
388 p0 S 0:00 _ su - root
389 p0 S 0:00 _ -sh
408 p0 R 0:00 _ ps af
[root@localhost /root]#
Connected To venezia [14.0.0.1]
VT320
NUM
15, 25
04:16

Figura 2. Salida de la ejecución del comando ps af.

Según los valores que se suministre en el parámetro *pid*, la llamada se comporta de la siguiente manera:

- *pid* > *0*: Se espera a la finalización del proceso cuyo identificador corresponde al parámetro.
  pid = - *1*: Se espera a la finalización del primer proceso hijo, por lo que resulta similar a *wait*.
- *pid* = *0*: Espera a la terminación de todos los procesos hijos que pertenecen al mismo grupo de procesos que el invocante.
- *pid* < *-1*: Espera a la finalización de todos los procesos hijos cuyo identificador de grupo es igual al valor absoluto del parámetro.

Esta llamada devuelve dos valores:

- El código de retorno que puede ser:
  - El *PID* del proceso que ha finalizado.
  - *1* en caso de fallo.
  - El estado del proceso hijo en la variable en el campo apuntado por el segundo parámetro.

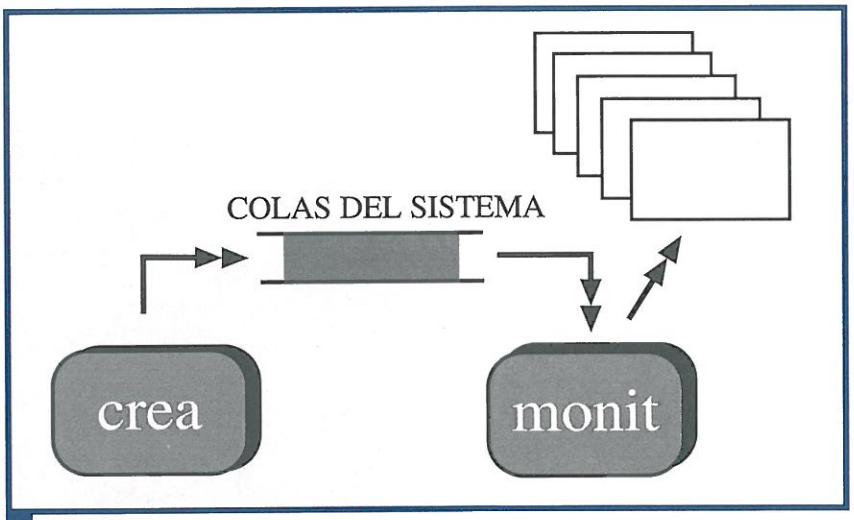


Figura 3. Esquema de los programas que intervienen en el ejemplo.

LOS DEMONIOS

Se trata de unos procesos que se caracterizan porque se ejecutan haciendo que su proceso padre sea el proceso *init*, cuyo identificador de proceso es el **1**, y no tienen como terminales asociados de entrada y salida la pantalla y el teclado respectivamente.

Si se arranca un proceso normal, bien desde la consola del sistema o un terminal *tty*, el terminal que ha lanzado el proceso queda bloqueado hasta que finalice dicho proceso, a no ser que este proceso se arranque en segundo plano (*background*). En cualquier caso, el proceso siempre queda ligado a la *shell* desde la que se ha arrancado por medio de una relación padre/hijo por un lado, y por otro al terminal desde el que se arrancó dicho proceso. Esta situación no es deseable para cierto tipo de procesos como pueden ser aquellos procesos que están manteniendo una sesión con otro proceso mediante un *socket TCP/IP* para controlar un dispositivo en tiempo real 24 horas al día, con lo que no es lógico tener un terminal *tty* bloqueado por esta razón, sin mencionar el peligro que puede suponer que alguien pulse las teclas **Ctrl-C** por error y finalice el proceso.

Un proceso para convertirse en demonio deben realizar dos tareas:

- Independizarse del padre y "engancharse" del *init*.
- Independizarse del terminal.

El siguiente código muestra como se realizan estas tareas:

```
if((pid  = fork()) < 0)
{
    perror("\nError en fork");
    exit(0);
}
else
    if (pid != 0)
    {
        sprintf(areaMensaje,
        "Saliendo proceso padre PID:
  %d",getpid());
        printf("%s\n",areaMensaje);
        escribirMensaje(areaMensaje);
        exit(0);
    }
setsid();
chdir(dirlog);
/* Lógica del programa */
exit(0);
```

El esquema general de este código ya ha sido explicado anteriormente y ahora tan sólo debemos observar que el proceso padre ya finaliza. Cuando un proceso ya ha finalizado todos los procesos hijos que se siguen ejecutando van a pasar a depender en el orden de la jerarquía del proceso número *1*, que no es otro que el proceso *init*.

La siguiente acción consiste en independizarse de los terminales, lo que se logra creando una nueva sesión para el proceso mediante la función *setsid*.

Este último paso, si bien no es necesario realizarlo para la creación de un demonio, resulta bastante conveniente con el fin de poder asegurarnos de que el proceso va a comenzar a trabajar en el directorio de trabajo deseado, y no desde el directorio en que se había situado la *shell* cuando se arrancó el proceso.

PLANIFICACIÓN DE PROCESOS

Una vez que se han visto las propiedades más notables de los procesos, así como la forma en que se pueden crear mediante llamadas al sistema, vamos a ver cómo gestiona el sistema operativo el acceso de los mismos al control del procesador. El encargado de realizar esta tarea es un componente del núcleo llamado *Scheduler* (Planificador).

Los procesos que están listos para poder ser ejecutados se colocan en una de las tres colas que tiene el Planificador. Dos de estas colas son para los llamados procesos en tiempo real y la otra para el resto de procesos, tales como los habituales de usuario.

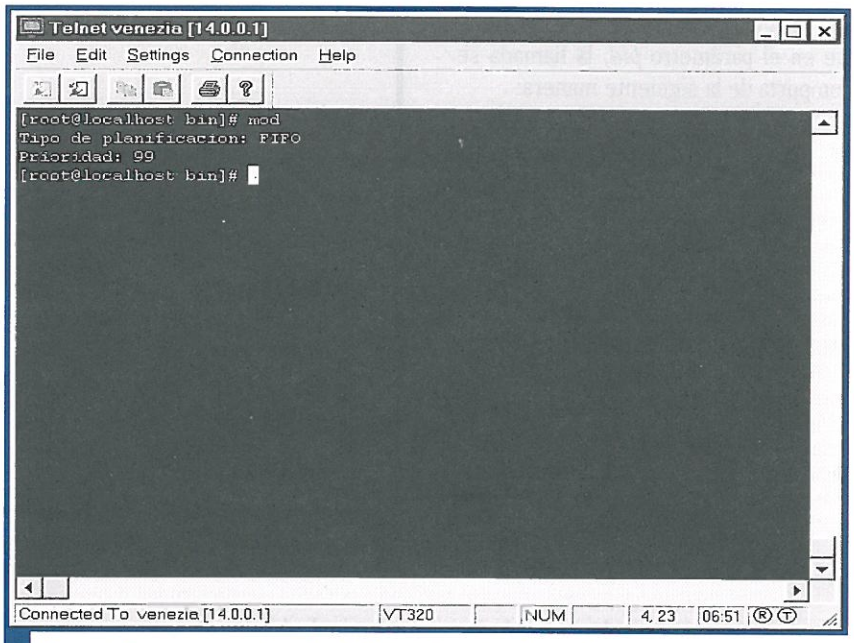


Figura 4. El programa mod pone la shell desde la cual se lanza el proceso.

En un sistema con un único procesador solamente puede haber un proceso en ejecución, mientras que los demás están en estado Preparado

Dentro de cada cola los procesos se colocan en un orden determinado en función de un valor que determina la prioridad, es decir en cada cola los procesos no se colocan según llegan, sino en función de un atributo del proceso que marca las prioridades. La prioridad de un proceso es un valor formado por otros dos, por un lado la propiedad fija que es adjudicada a cada proceso por medio de una llamada al sistema y la prioridad variable que se ajusta de manera dinámica.

La prioridad dinámica es gestionada por el sistema operativo en función de los recursos consumidos por el proceso, penalizando en alguna forma a aquellos procesos que consumen más recursos. También es posible modificarla desde la *shell* por medio del comando *nice*, aunque este tipo de prioridad está definida para los sistemas de planificación que no son de tiempo real. En este artículo nos vamos a centrar en la prioridad estática así como en las colas del planificador.

Tal y como se ha mencionado arriba existen tres colas para el planificador:

- *SCHED_FIFO*
- *SCHED_RR*
- *SCHED_OTHER*

La política del planificador es la siguiente:

- Se elige el proceso de la cola *SCHED_*FIFO de más alta prioridad y se ejecuta a menos que se de una de las siguientes condiciones:
- Existe otro proceso en dicha cola de más alta prioridad.
- El proceso genera una interrupción.
- Si no existen procesos en la cola *SCHED_FIFO* se selecciona un proceso de la cola *SCHED_RR*, y se ejecuta y no se interrumpe hasta que se dé una de las siguiente condiciones:
- Existe un proceso de cualquier prioridad en la cola *SCHED_FIFO*.
- Existe un proceso de prioridad igual o superior en la cola *SCHED_RR*.
- El proceso genera una interrupción.
- Si no existen procesos en las colas *SCHED_FIFO* y *SCHED_RR* se selecciona el proceso de más alta prioridad de la cola *SCHED_OTHER*, y sólo se interrumpe cuando exista un proceso de más alta prioridad en cualquiera de las tres colas.

Las llamadas al sistema que permiten modificar la política de planificación y las prioridades de un proceso son:

```
int sched_setscheduler (pid_t pid, int politica,
                        const struct
    sched_param *parametros);

int sched_getscheduler (pid_t pid);
```

siendo los parámetros que las definen los siguientes:

pid: identificador del proceso al que se desea cambiar la prioridad.

politica: es un valor entero que puede admitir los valores: *SCHED_FIFO (1)*, *SCHED_RR (2)* ó *SCHED_OTHER (0)*.

parametros: se trata de una estructura sobre la que se pueden modificar otros parámetros de la planificación del proceso.

Los prototipos de las funciones que gestionan la prioridad estática son:

```
int setpriority (int grupo, int ident, int prio);
int getpriority (int grupo, int ident);
```

EJEMPLO PRÁCTICO

Para poder ver un efecto práctico de lo se ha explicado en estas páginas se adjuntan tres programas: **monit**, **mod** y **crea**, siendo sus fuentes respectivos: **monit.c**, **mod.c** y **crea.c**. Además hay un fichero *include* con los valores que requieren los programas.

En el fichero *Makefile* se encuentran las sentencias para compilars, lo que se realiza con el comando: **make instala**. Esta sentencia compila y enlaza los tres programas y además crea los ficheros **semkey** y **mq.log** en el directorio */tmp*. Si se desea cambiar estos nombres además de modificarlos en el *Makefile* será necesario cambiar los nombres de los ficheros en el fichero **valores.h**. Si en vez de **make instala** se ejecuta **make todo** se compilan y enlazan los tres programas.

> El código de retorno de la llamada al sistema fork permite saber si se está en el proceso padre o en el hijo

En la figura 3 se puede observar un diagrama de bloques de los programas, siendo el principal **monit**, que se convierte en demonio una vez ha sido arrancado mediante la sentencia del mismo nombre. Un tema que tener en cuenta consiste en cómo se puede realizar la comunicación con este proceso una vez que se ha convertido en demonio, ya que una de las características de los demonios es su independencia del teclado. Las repuestas son varias: colas de mensajes, *sockets*, *pipes* con nombres, de por sí los mecanismos de comunicación entre programas justificarían varios artículos, únicamente decir que el programa utiliza para comunicarse las colas de mensajes. Los mensajes de salida realizan formateados sobre el fichero */tmp/mq.log*, que puede ser cambiado tal y como se indica en las instrucciones que vienen en el fichero *Makefile*.

El programa una vez ha inicializado la cola de mensajes y antes de convertirse en demonio se establece con la política de prioridad *SCHED_FIFO y* con la máxima prioridad.

El programa **crea**, utilizando la cola de mensajes da una orden al programa **monit** para que cree un nuevo proceso (haciendo uso de la llamada al sistema *fork*), y según el parámetro que reciba le asigna una de las tres políticas de planificación. La sintaxis del comando **crea** es la siguiente:

```
crea [fifo | rr | otro ]
```

Los procesos son creados con una prioridad de **2** y son bucles simples.

Una vez que se ha arrancado el demonio *monit* ya estamos en condiciones de empezar a crear procesos y ver cómo se van ejecutando. Pero antes de proceder hay que plantearse la siguiente pregunta: si se arranca un proceso que tiene un bucle y además está en la cola *SCHED_FIFO*, ¿cómo afecta esto a la *shell* desde la que estoy trabajando?. La respuesta se resume en una palabra: fatal. Si una vez arrancado *monit* se crea un proceso mediante los comandos: **crea rr**, o **crea fifo**, la *shell* que está en la cola *SCHED_OTHER* difícilmente podrá acceder a la *CPU* para ejecutarse.

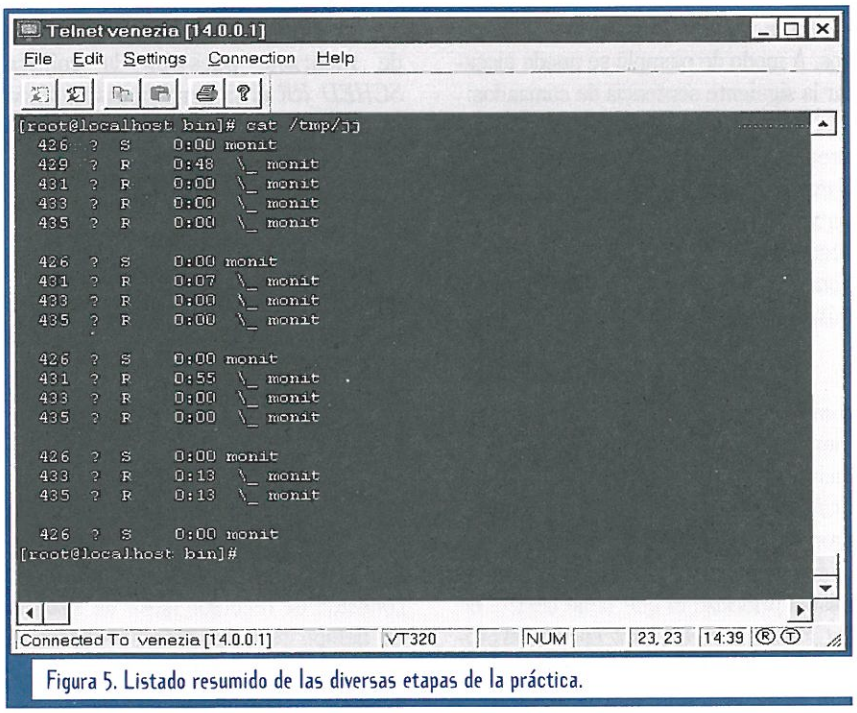

Telnet venezia [14.0.0.1]
File Edit Settings Connection Help
[root@localhost bin]# cat /tmp/jj
426 ? S 0:00 monit
429 ? R 0:48 _ monit
431 ? R 0:00 _ monit
433 ? R 0:00 _ monit
435 ? R 0:00 _ monit
426 ? S 0:00 monit
431 ? R 0:07 _ monit
433 ? R 0:00 _ monit
435 ? R 0:00 _ monit
426 ? S 0:00 monit
431 ? R 0:55 _ monit
433 ? R 0:00 _ monit
435 ? R 0:00 _ monit
426 ? S 0:00 monit
433 ? R 0:13 _ monit
435 ? R 0:13 _ monit
426 ? S 0:00 monit
[root@localhost bin]#
Connected To venezia [14.0.0.1]
VT320
NUM
23, 23
14:39

Figura 5. Listado resumido de las diversas etapas de la práctica.

Conclusión: todas las *shell* se quedan bloqueadas. El autor de este artículo no ha encontrado otra solución que pulsar el botón de *reset* del equipo para salir de esta situación.

La solución para evitar esta situación pasa por poner al menos una *shell* en la cola *SCHED_FIFO* con más prioridad que los procesos que se van a arrancar. El programa **mod** pone la *shell* en que se ejecuta en dicha cola con una prioridad de **99**.

El atributo que define unívocamente a cada proceso es el identificador de proceso (o process id), más conocido por PID

Las llamadas al sistema que realiza este programa y que están relacionadas con la gestión de procesos son: *getppid* para obtener el identificador del proceso padre, que es el que interesa cambiar de política de planificación. Es decir, hay que obtener el identificador de la *shell* que ejecuta el programa **mod**, ya que este programa es en sí mismo un proceso y una vez obtenido el identificador cambiar la política y prioridad por medio de la función *sched_setscheduler*.

El propio programa, si todo va correctamente, indica con unos mensajes la nueva prioridad y planificación de la *shell*. Además todo esto debe realizarse desde las pantallas locales y en modo texto puro, pues ciertos procesos de gestión del entorno *X*, así como de comunicaciones pueden ejecutarse en una política de prioridades diferente y quedar por debajo de las rutinas de **monit**.

Una buena configuración para poder trabajar con las rutinas consiste en tener varias sesiones en los terminales virtuales (*Alt+Fn*), dejando la *shell Alt+F1* con la máxima prioridad y ejecutar la sentencia **crea fifo**, cambiar a otra sesión *Alt+Fn* (n>1), y observar como las demás sesiones han quedado congeladas, como si el sistema estuviera colgado. Pero desde la sesión *Alt+F1* se puede trabajar normalmente, siendo posible comprobar como trabaja el proceso con el comando **ps xf | grep monit** (El parámetro *x* del comando *ps* lista los procesos demonios). Si ejecutamos damos varios de estos comandos podemos observar como el tiempo de *CPU* aumenta considerablemente. Matando el proceso, se puede acceder a las demás sesiones y ver como estas funcionan normalmente e incluso han respondido a las peticiones pendientes. Lo que parecía un "cuelgue" del sistema, no era más que una planificación de tareas poco acertada, y únicamente nos ha salvado de tener que hacer un al equipo la precaución de haber cambiado la prioridad de una *shell*. Si no se hubiera tomado esta precaución, todas las apariencias: procesos bloqueados, la luz de acceso al bus de datos apagada, etc., apuntan a un "cuelgue".

Una vez que se ha trabajado con este ejemplo y todo se ha realizado correctamente (fácil de comprobar, pues si todo va bien no hay que pulsar el botón *reset)*, se pueden arrancar varios procesos. A modo de ejemplo se puede ejecutar la siguiente sentencia de comandos:

crea fifo
Tomar nota del número de proceso con:
ps xf | grep monit
crea fifo
Tomar nota del número de proceso de la misma forma.

En este punto se puede observar como solamente está trabajando el primer proceso. Esto es coherente con lo dicho anteriormente: un proceso en *SCHED_FIFO* sólo cede la *CPU* a otro proceso *SCHED_FIFO* con más prioridad; como ambos procesos tienen la misma prioridad el que tiene control de la *CPU* no la suelta, a excepción del proceso *monit* padre y nuestra propia *shell* que tiene una prioridad superior y la misma política de planificación.

A continuación se ejecutan los siguientes comandos:

crea rr
Tomar nota del número de proceso.
crea rr
Tomar nota del número de proceso.

Observamos la evolución de consumos de procesos y la situación sigue igual ya que los procesos en *SCHED_RR* se ejecutan después de los procesos en *SCHED_FIFO*.

Lo siguiente que hacemos es matar el primer proceso *monit,* observamos los recursos y vemos que el siguiente proceso con *SCHED_FIFO* toma control la *CPU* sobre los otros dos procesos *SCHED_RR*.

Si ahora arrancamos otro proceso *SCHED_FIFO* mediante **crea fifo** y matamos el que está activo, el nuevo proceso toma el control de la *CPU* y los procesos *SCHED_RR* siguen parados. Por último mataremos el proceso recién creado y observaremos la evolución de los consumos de los dos procesos que quedan en máquina. En este caso ambos progresan en el consumo de recursos, pues en la política *SCHED_RR* el sistema va balanceando el consumo de recursos entre los procesos con igual prioridad.

CONCLUSIÓN

A lo largo de este artículo se ha visto de forma general el mecanismo de prioridades de Linux. Comprendiendo y utilizando bien estos mecanismos es posible convertir un equipo Linux desde una máquina con usuarios en tiempo compartido y una política de penalización por consumo de recursos hasta un sistema de tiempo real con proceso críticos, e incluso hacer cohabitar ambos tipos de sistemas en una misma máquina.

GLIDE (II)

Constantino Sánchez Ballesteros (constantino@nexo.es)

Como no podía ser de otro modo, en esta nueva entrega procederemos a crear nuestro primer polígono en Glide. Utilizaremos un sombreado Gouraud para los vértices. Además de ésto, también abordaremos los pasos más necesarios para construir un Z buffer.

CLIPPING

El hardware de la *Voodoo Graphics* soporta *per-pixel clipping* (recorte de polígonos por pixel) a un rectángulo arbitrario mediante la función de *Glide grClipWindow()*. De esta forma, cualquier pixel que se encuentre fuera de la ventana de *clipping* será desechado. Esta ventana también especifica el área que borrará la función *grBufferClear()*.

La forma de definirla consiste en utilizar dos pares de *integer* en el rango **[0..1024]** especificando las esquinas izquierda y derecha e inferior y superior de las esquinas del rectángulo:

La función *grClipWindow()* tiene cuatro parámetros que definen el rectángulo de *clipping*. Los valores deben ser menores o iguales a la resolución de la pantalla y mayores o iguales que **0**, ya que de otro modo se ignorará. *Glide* no utiliza *clipping* fuera de la ventana que contiene la aplicación. Para un óptimo rendimiento, una aplicación debería realizar el *clipping* antes de pasar cualquier primitiva a *Glide*.

```
void grClipWindow (FxU32 minX, FxU32 minY,
      FxU32 maxX, FxU32 maxY )
```

La función grClipWindow() tiene cuatro parámetros que definen el rectángulo de clipping

Los valores por defecto de la ventana de *clip* corresponden al tamaño de resolución de pantalla:

(0,0,640,480) para 640¥480

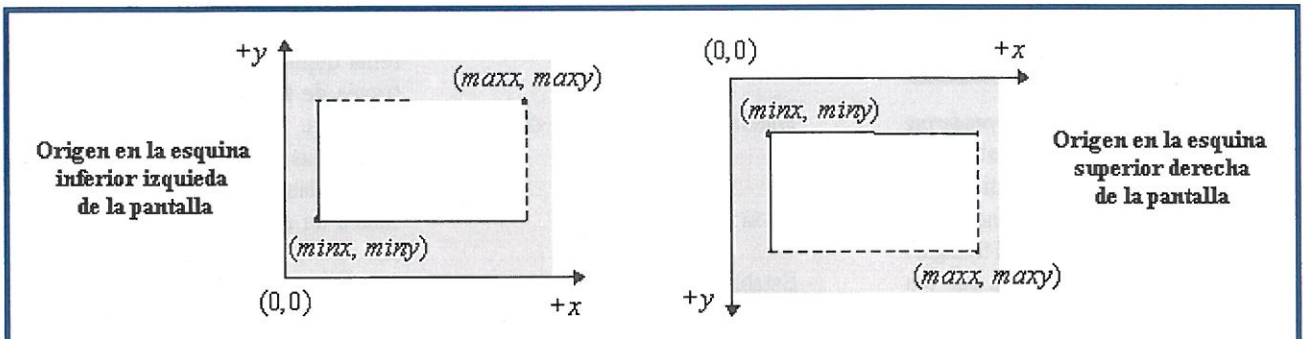


Figura 1. Clipping en Glide.

(0,0,800,600) para 800¥600

Para desactivar el *clipping* simplemente establecemos el tamaño de la ventana de clip al tamaño de la pantalla.

La función grClipWindow permite efectuar Clipping sobre las primitivas representadas en la pantalla

TRIÁNGULOS

El triángulo es la primitiva básica de *Glide*. La función *grDrawTriangle()* renderiza un triángulo orientado con los vértices *a*, *b*, y *c* en la pantalla.

```
void grDrawTriangle (const GrVertex *a, const
    GrVertex *b, const GrVertex *c )
```

Los triángulos son renderizados con las siguientes reglas de relleno:

- Triángulos de área igual a cero renderizan cero pixels.
- Los Pixels son renderizados si su centro está dentro de los límites del triángulo

CLIPPING AL TRIÁNGULO

Un pixel utilizará proceso de *rendering* siempre que se escriba al *Frame Buffer*. Una solución alternativa para aumentar el rendimiento de renderizado consiste en efectuar *clipping* al triángulo y procesar solamente los pixels que van a ser visualizados. La librería de utilidades de Glide tiene la siguiente función específica para realizar esta tarea:

```
guDrawTriangleWithClip()
```

Esta función aplica *clipping* al triángulo del rectángulo especificado por *grClipWindow()* y dibuja el polígono resultante.

```
void guDrawTriangleWithClip (const GrVertex
    *a, const GrVertex *b,
    const GrVertex *c )
```

PUNTOS

La función *grDrawPoint()* renderiza un simple punto en la pantalla, el cual será tratado como un triángulo con los vértices cercanos coincidentes (es decir, un pequeñísimo triángulo). Si se renderizarán multitud de puntos, se notaría una gran disminución de la velocidad de *render* cuando se escriben los pixels al *frame buffer*. *grDrawPoint()* envía tres vértices por punto al hardware y repite el proceso en las tres esquinas.

```
void grDrawPoint (const GrVertex *a )
```

El siguiente fragmento de código borra la pantalla utilizando el color negro y dibuja mil puntos al azar. Por defecto, el *buffer* de *rendering* se establece con el valor *GR_BUFFER_ BACKBUFFER* y el color del *buffer* como "escritura". El color blanco se realiza especificando valores máximos para el rojo, verde y azul. Algunos puntos tendrán que utilizar la técnica de *clipping* ya que se saldrán de la pantalla.

```
int n;
GrVertex p;
```

Ponemos el *backbuffer* de color negro:

```
grBufferClear(0, 0, 0);
```

Establecemos el color blanco:

```
GrColorCombine (GR_COMBINE_FUNC-
    TION_LOCAL, GR_COMBINE_FAC-
    TOR_NONE,
    GR_COMBINE_LOCAL_CONSTANT,
    GR_COMBINE_OTHER_NONE,
    FXFALSE);
grConstantColorValue(~0)
```

Generamos y dibujamos mil puntos al azar:

```
for (n=0; n<1000; n++) {
    p.x = (float) (rand() % 1024);
    p.y = (float) (rand() % 1024);
    grDrawPoint(p);
 }
```

LÍNEAS

La función *grDrawLine()* renderiza y construye una línea en la pantalla. Si están activados los efectos especiales (por ejemplo *fog*, *blending*, *chroma-key*, *dithering*, etc.) afectará a la apariencia de dicha línea.

```
void grDrawLine (const GrVertex *a, const
    GrVertex *b )
```

SOMBREADO GOURAUD

Mediante el sombreado *Gouraud* logramos que un objeto 3D tenga un aspecto suavizado en las aristas en lugar de poligonal (sombreado *Flat*). En el modo de sombreado *Flat*, el sistema duplica el color de un vértice a través de todas las demás caras de la primitiva. En los modos *Gouraud* y *Phong*, las Normales de los vértices son usadas para dar un aspecto suavizado a un objeto poligonal.

En el sombreado *Gouraud*, el color y la intensidad de los vértices adyacentes se obtiene mediante interpolación a través del espacio que los

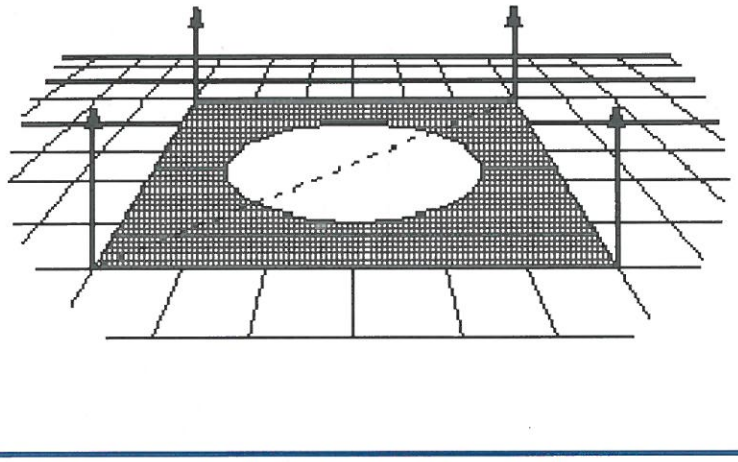


Figura 2. Diferencias entre la aplicación del sombreado Gouraud y el Phong.

separa. En el sombreado *Phong* (no soportado actualmente soportado por *Direct3D*), el sistema calcula el valor de sombreado apropiado por cada *pixel* que se encuentre en una cara. La mayoría de las aplicaciones utilizan el sombreado *Gouraud* porque permite obtener objetos con una apariencia muy suave y un consumo de *CPU* relativamente bajo.

En el modo de sombreado Flat, el sistema duplica el color de un vértice a través de todas las demás caras de la primitiva

Por ejemplo, *Gouraud* y *Phong* producirían unos resultados muy diferentes en el caso mostrado en la figura 2, en el cual, una luz de tipo *Spotlight* está completamente contenida en una cara:

En este caso, el sombreado *Phong* calcularía el valor de cada *pixel* y visualizaría la luz (*Spotlight*). El modo *Gouraud*, el cual efectúa interpolación entre vértices, omitiría la luz completamente y la cara sería renderizada desperdiciando la luz.

En el modo *Flat*, la siguiente pirámide sería visualizada con bordes afilados entre las uniones de las caras; el sistema generaría Normales de cara automáticas. En los modos *Gouraud* y *Phong*, los valores de sombreado deberían ser interpolados a través de los bordes y su apariencia final sería una superficie curvada:

Si utilizamos el sombreado *Gouraud* para visualizar superficies curvas e incluir objetos con bordes afilados, nuestra aplicación necesitaría duplicar las Normales de los vértices en cualquier intersección de caras donde el borde afilado se requiriese, tal y como se muestra en la figura 4:

Además de obtener en un mismo objeto superficies curvadas y afiladas, el modo *Gouraud* visualiza las superficies *Flat* de forma más realista. Una cara en el modo *Flat* es un color uniforme, mientras que con *Gouraud* permite el fundido de la luz entre caras de forma real.

Vamos a comenzar la programación de un simple triángulo en *Glide* con sombreado *Gouraud*. El primer paso que se debe efectuar en cualquier programa en C consiste en incluir las librerías necesarias para que funcione el proyecto.

Recomiendo al lector que eche un vistazo al artículo anterior de esta serie para saber cómo incluir en nuestro pro-

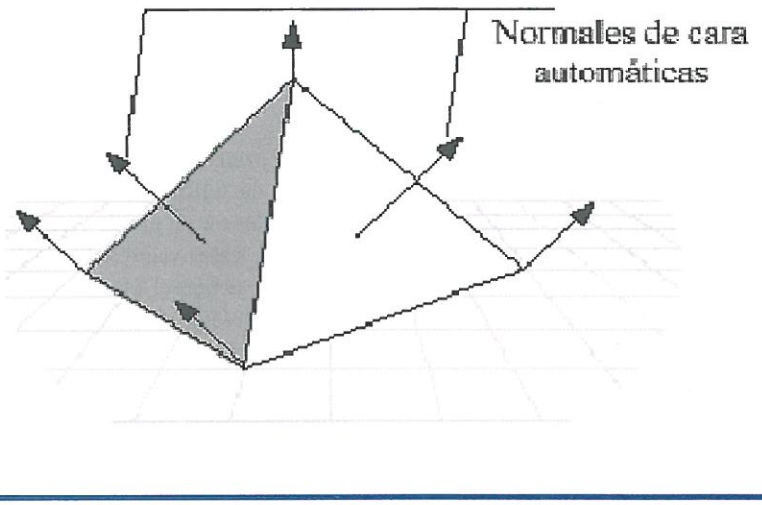


Figura 3. Sombreado de tipo Flat con detalles de las normales.

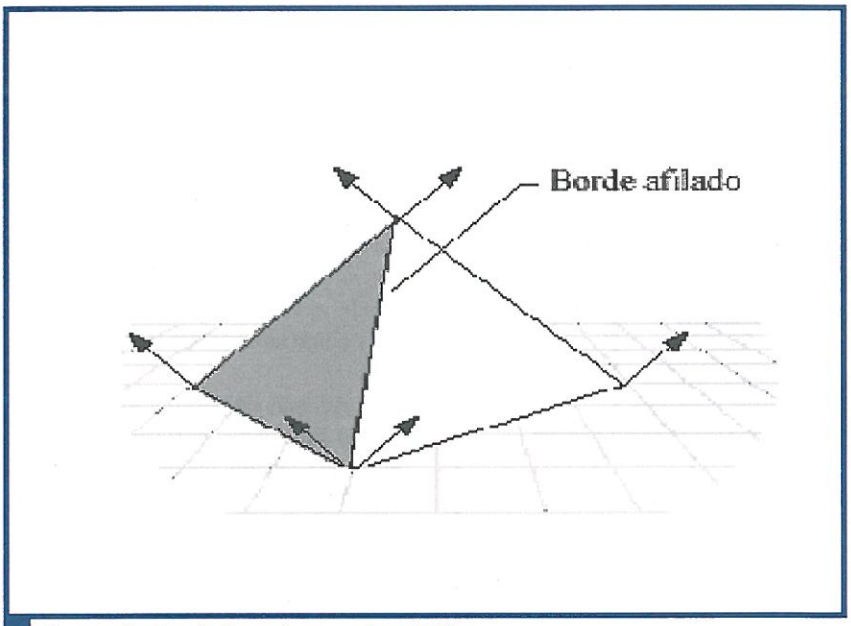


Figura 4. Sombreado Gouraud con visualización Flat.

yecto las librerías de *Glide* y los parámetros necesarios que se deben activar en la pestaña *Settings* de *Visual C*:

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <assert.h>
#include <string.h>
#include <glide.h>
#include "tlib.h"
```

Las tarjetas Voodoo 2 sólo permiten resoluciones de pantalla de 640x480 / 800x600 pixels

Creamos una variable de tipo *GrHwConfiguration* para establecer la configuración de la *Voodoo* en tiempo de ejecución:

```
GrHwConfiguration hwconfig;
static char version[80];
```

Las siguientes variables mostrarán un texto en la consola de la aplicación para definir lo que va a realizar el programa.

```
static const char name[]    = "Gouraud";
static const char purpose[] = "Dibuja un triángulo con sombreado Gouraud";
static const char usage[]   = "-n <frames> -r <res> -d <filename>";
```

En el siguiente procedimiento *(main)* se establece todo el código principal del programa. Os recuerdo que en la programación de *Glide* no es necesario crear una ventana manualmente ni utilizar las *MFC* de *Microsoft*:

```
void main( int argc, char **argv)
{
 char match;
 char **remArgs;
 int  rv;
```

Utilizaremos una resolución de pantalla de **640x480**. Por ello, debemos establecer un par de variables que contengan estos valores de resolución, además de definir el *flag GR_RESOLUTION_640x480*, necesario para activar dicha resolución:

```
GrScreenResolution_t resolution = GR_RESOLUTION_640x480;
float          scrWidth  = 640.0f;
float          scrHeight = 480.0f;
int frames                = -1;
FxBool         scrgrab = FXFALSE;
char           filename[256];
```

El programa se configurará para leer los argumentos introducidos desde *MS-DOS* al ejecutar el programa (tipos de resolución). La ejecución funcionará habiendo o no argumentos, pero es una funcionalidad adicional implementada:

```
while( rv = tlGetOpt( argc, argv, "nrd", &match,
       &remArgs ) )
{
 if ( rv == -1 ) {
   printf( "argumentos no reconocidos" );
   printf( "%s %s\n", name, usage );
   printf( "Resoluciones disponibles:\n%s\n",
     tlGetResolutionList() );
   return;
 }

 switch( match ) {
  case 'n':
   frames = atoi( remArgs[0] );
   break;
  case 'r':
   resolution = tlGetResolutionConstant(
     remArgs[0],
                &scrWidth, &scrHeight );
   break;
  case 'd':
   scrgrab = FXTRUE;
   frames = 1;
   strcpy(filename, remArgs[0]);
   break;
  }
}
```

Establecemos la resolución de la pantalla mediante los valores contenidos en las variables **scrWidth** y **scrHeight**:

```
tlSetScreen( scrWidth, scrHeight );
```

Obtenemos la versión de *Glide* instalada en nuestro equipo. Actualmente se encuentra disponible la *3.0*:

```
grGlideGetVersion (version );
```

Imprimimos en la consola el resultado de la versión obtenida, así

como la resolución que se utilizará en el programa:

```
printf( "%s:\n%s\n", name, purpose );
printf( "%s\n", version );
printf( "Resolution: %s\n",
      tlGetResolutionString( resolution ) );

if ( frames == -1 ) {
   printf( "Inicializando Glide" );

}
```

Una vez obtenida toda la información necesaria, inicializamos *Glide* mediante *grGlideInit*:

```
grGlideInit();

assert( grSstQueryHardware( &hwconfig ) );
```

Seleccionamos la primera tarjeta instalada en nuestro equipo (por si tenemos dos conectadas en *SLI*):

```
GrSstSelect(0);
```

Creamos la ventana que contendrá la aplicación (*grSstWinOpen*). En sus parámetros se debe incluir la resolución, el refresco de pantalla, el formato de color y la posición inicial sobre la que *Glide* comenzará a dibujar pixels; en nuestro caso, la esquina superior izquierda de la pantalla (*GR_ORIGIN_UPPER_LEFT*):

```
assert( grSstWinOpen( 0, resolution,
      GR_REFRESH_60Hz,
GR_COLORFORMAT_ABGR,
      GR_ORIGIN_UPPER_LEFT,
            2, 1 ) );

tlConSet( 0.0f, 0.0f, 1.0f, 1.0f, 60, 30, 0xffffff );
```

Mediante **grColorCombine** establecemos el estado del *render*; en nuestro caso sombreado *Gouraud*:

```
grColorCombine( GR_COMBINE_FUNC-
      TION_LOCAL,
GR_COMBINE_FACTOR_NONE,
      GR_COMBINE_LOCAL_ITERATED,
         GR_COMBINE_OTHER_NONE,
         FXFALSE );

tlConOutput( "Presione una tecla para salir" );
```

A partir de este punto comenzamos el bucle que se encargará de dibujar el triángulo. Es necesario crear tres variables de tipo *GrVertex* para definir los tres vértices que definirán el triángulo que se va a representar en pantalla. Estas tres variables son **vtxA**, **vtxB** y **vtxC**:

```
While( frames— && tlOkToRender())
{
   GrVertex vtxA, vtxB, vtxC;

   if (hwconfig.SSTs[0].type == GR_SSTTY-
      PE_SST96) {
    tlGetDimsByConst(resolution, &scrWidth,
      &scrHeight );
    grClipWindow(0, 0, (FxU32) scrWidth,
      (FxU32) scrHeight);
   }
```

Limpiamos el *buffer* que contendrá los gráficos mediante el color negro (valor **0**):

```
grBufferClear( 0x00, 0, GR_WDEPTHVA-
      LUE_FARTHEST );
```

En este punto del programa realizamos uno de los pasos más importantes: la creación del triángulo que veremos en pantalla. Para ello es totalmente necesario definir el valor de cada vértice que conformará el triángulo final.

En la tabla 1 se pueden apreciar los campos que componen la estructura *GrVertex*, mediante la que se define un vértice.

Las coordenadas **x** e **y** son valores en coma flotante de 32 *bits* que representan la posición del vértice en el espacio de la pantalla. Mientras que el hardware de la *Voodoo* sólo renderiza triángulos, *Glide* permite funciones para dibujar puntos, líneas, polígonos y triángulos.

En los modos Gouraud y Phong, los valores de sombreado deberían ser interpolados a través de los bordes

Cuando un punto, línea, triángulo o polígono es renderizado, su apariencia reflejará el estado actual del canal de *render* de la tarjeta gráfica. Por ejemplo, si *texture mapping* se encuentra activado (mapeado de texturas), el punto, línea o polígono aparecerá texturizado. De forma similar, *alpha blending*, *fogging* (niebla), color, efectos de luces, *chroma-keying*, y otros efectos especiales contribuirán a definir la apariencia de cualquier forma geométrica dibujada en pantalla.

Continuando con el código, primero definimos la posición del vértice en la pantalla (propiedades *X* e *Y*) y estableceremos su color mediante las propiedades *R,G,B* que representan los colores rojo, verde y azul.

```
vtxA.x = tlScaleX( 0.3f ), vtxA.y = tlScaleY(
      0.3f );
```

Tabla 1. Estructura de GrVertex

```
typedef struct
{
 float x, y, z; " x, y, z del espacio de la pantalla. Z es ignorada"
 float ooz;      " 65535/z (utilizada para Z buffering) "
 float oow;      " 1/w (utilizada para W buffering) "
 float r, g, b, a;" Color RGB y alpha ([0..255.0]) "
 GrTmuVertex tmuvtx[GLIDE_NUM_TMU];
} GrVertex;
```

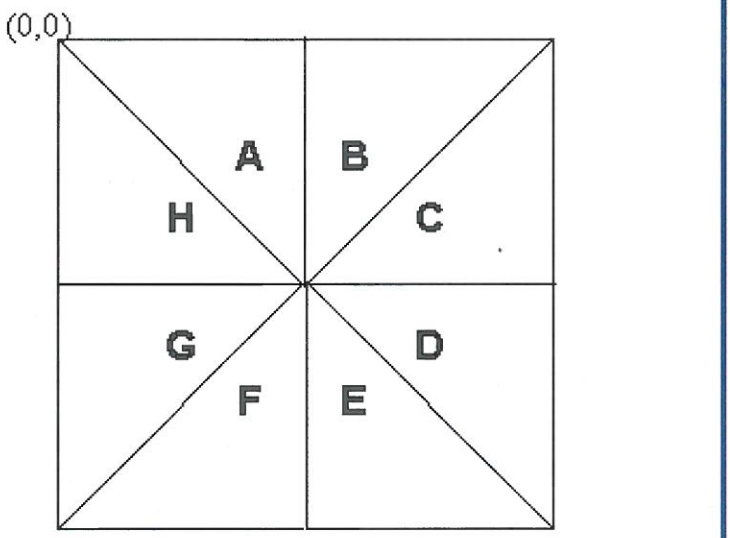


Figura 5. El pixel que une estos 8 triángulos sería renderizado en Glide por el triángulo D.

```
vtxA.r = 255.0f, vtxA.g = 0.0f, vtxA.b = 0.0f;

vtxB.x = tlScaleX( 0.8f ), vtxB.y = tlScaleY(
      0.4f );
vtxB.r = 0.0f, vtxB.g = 255.0f, vtxB.b = 0.0f;

vtxC.x = tlScaleX( 0.5f ), vtxC.y = tlScaleY(
      0.8f );
vtxC.r = 0.0f, vtxC.g = 0.0f, vtxC.b = 255.0f;
```

Finalmente, sólo nos queda dibujar el triángulo obtenido mediante la función de *Glide grDrawTriangle*:

```
grDrawTriangle( &vtxA, &vtxB, &vtxC );
```

La función anterior dibuja un triángulo y su estructura es la siguiente:

```
void grDrawTriangle( const GrVertex *a, const
      GrVertex *b, const GrVertex *c )
```

donde sus parámetros *a, b, c* definen la localización y atributos de los vértices que definen el triángulo.

Esta función renderiza un triángulo en la pantalla. Todos los atributos establecidos en *Glide* afectarán la apariencia del triángulo. Los triángulos son renderizados con las siguientes reglas para su relleno:

- Los triángulos con área cero renderizan cero *pixels*.
- Los *pixels* son renderizados sólo si están dentro del triángulo.

El centro de un *pixel* estará en un triángulo si se encuentra entre los tres vértices que definen el polígono. Si el *pixel* se encuentra fuera de los límites de los vértices no se dibujará.

En la figura 5, un *pixel* cuyo centro se encuentra en la intersección de ocho triángulos es renderizado sólo por el triángulo **D**:

Estas reglas de relleno garantizan que los polígonos sean representados en pantalla de forma correcta y sólo una vez.

Para finalizar con el programa, sólo nos queda volcar el contenido del *frame buffer* a la pantalla, para lo cual utilizaremos la función *grBufferSwap*:

```
 tlConRender();
 grBufferSwap( 1 );
}
```

Una vez acabado el programa debemos reiniciar *Glide* y la tarjeta mediante la función *grGlideShutdown*.

```
 grGlideShutdown();
 return;
}
```

Z BUFFERING

En el siguiente extracto de programa (el cual podréis encontrar completo en el *CD-ROM* que acompaña a la revista) se muestra la total creación de un *Z Buffer*. Para conseguir este proceso se crearán tres triángulos y se interconectarán entre sí en la pantalla demostrando la correcta visualización de los mismos sin ocultar un polígono al otro.

Creamos el *Z Buffer*:

```
grDepthBufferMode(
      GR_DEPTHBUFFER_ZBUFFER );
grDepthBufferFunction( GR_CMP_GREATER
      );
grDepthMask( FXTRUE );
```

Creamos variables para asignar los vértices y la profundidad del eje Z (**zDist**):

```
GrVertex vtxA, vtxB, vtxC;
float zDist;
```

Creamos una ventana de *clipping* para los polígonos y borramos el *buffer* de dibujado:

```
if (hwconfig.SSTs[0].type ==
      GR_SSTTYPE_SST96)
 {
  tlGetDimsByConst(resolution, &scrWidth,
      &scrHeight );
  grClipWindow(0, 0, (FxU32) scrWidth,
      (FxU32) scrHeight);
 }

grBufferClear( 0x00, 0,
      GR_ZDEPTHVALUE_FARTHEST );
```

Definimos los valores de los vértices para el primer triángulo:

```
vtxA.x = tlScaleX( 0.25f ), vtxA.y = tlScaleY( 0.21f );
vtxB.x = tlScaleX( 0.75f ), vtxB.y = tlScaleY( 0.21f );
vtxC.x = tlScaleX( 0.5f  ), vtxC.y = tlScaleY( 0.79f );
```

Los valores de profundidad del triángulo deben ser escalados de **0** a **65535.0** (*ooz = (1.0f / Z) * 65535.0f = 65535.0f / Z*) :

```
zDist = 10.0f;
vtxA.ooz = vtxB.ooz = vtxC.ooz = ( 65535.0f / zDist );
```

Pintamos de un color constante el triángulo y lo dibujamos en el *buffer* correspondiente mediante *grDraw Triangle*:

```
grConstantColorValue( 0x00808080 );

grDrawTriangle( &vtxA, &vtxB, &vtxC );
```

Realizamos el mismo proceso del primer triángulo pero aplicado al segundo que se representará en la pantalla. Primero, definimos vértices:

```
vtxA.x = tlScaleX( 0.86f ), vtxA.y = tlScaleY( 0.21f );
vtxB.x = tlScaleX( 0.86f ), vtxB.y = tlScaleY( 0.79f );
vtxC.x = tlScaleX( 0.14f ), vtxC.y = tlScaleY( 0.5f );
```

Establecemos valores de profundidad de los vértices:

```
zDist = 12.5f;
vtxA.ooz = vtxB.ooz = ( 65535.0f / zDist );
zDist = 7.5f;
vtxC.ooz = ( 65535.0f / zDist );
```

Dibujamos el triángulo con un color constante pero diferente al primer triángulo para observar mejor el resultado final visualizado en la pantalla:

```
grConstantColorValue( 0x0000FF00 );

grDrawTriangle( &vtxA, &vtxB, &vtxC );
```

CONCLUSIÓN

A lo largo de este capítulo se ha profundizado en las características y en la programación mediante *Glide*. El código de los ejemplos comienza a tener cierta entidad y se van apreciando los resultados y las posibilidades que ofrece. En el próximo artículo avanzaremos hacia algunas de las opciones más interesantes que tenemos a nuestro alcance, utilizando la potencia y todas las prestaciones que *Glide* nos ofrece.

Creación de un buscador Web (II)

Enrique de la Lastra (elastra@redestb.es)

El corazón de un buscador Web es un programa denominado Robot. Éste se encarga de analizar todos los posibles enlaces de una página HTML, obtener algún tipo de información sobre las páginas y archivos en ella referidos, y continuar analizando de forma recursiva los datos enlazados.

INTRODUCCIÓN

Un Robot es un programa que realiza peticiones HTTP de la misma manera que lo hace un Navegador Web, pero a diferencia de éste, lo ejecuta de forma automática siguiendo unos criterios de actuación predefinidos. Es decir, un Robot automatiza – de ahí su nombre - el proceso de petición de páginas HTML o recursos Web en general. Es decir, lanza una petición HTTP de una página y en la respuesta busca enlaces a distintos recursos; cuando estos enlaces corresponden a otras páginas HTML lanza nuevas peticiones sobre estas páginas y así sucesivamente.

Los Robots asociados a los buscadores añaden básicamente una sola función al proceso descrito: guardar algún tipo de información de las páginas que van visitando.

En este artículo vamos a dar los primeros pasos necesarios para implementar un Robot Web. Este Robot, a partir de una URL (*Uniform Resource Locator,* Localizador Uniforme de Recursos), visita la página correspondiente a esa URL, la indexa y lanza peticiones a todas las páginas referidas en los enlaces contenidos en esa primera página. A su vez, vuelve a lanzar peticiones sobre los enlaces de las nuevas páginas, y así una y otra vez, en un principio hasta el infinito (hasta la última página Web) o hasta que no se encuentren más enlaces.

Se puede limitar la búsqueda de páginas hasta un límite de saltos desde la primera página o simplemente mediante un temporizador. Por supuesto también habrá que disponer de un mecanismo manual de parada, que deberá guardar el estado en que se encuentre el análisis, de forma que pueda continuar donde se haya parado. Recordamos que un Robot no es una especie de "gusano" que se introduce en los servidores Web y viaja de Servidor en Servidor husmeando entre las páginas HTML, sino que es un programa que se ejecuta localmente en nuestra máquina, realizando peticiones HTTP y por tanto "gastando" una conexión a Internet.

PRIMEROS PASOS

La primera pregunta que se nos ocurre a la hora de plantear un Robot Web es la siguiente: ¿Cómo decidirá nuestro robot qué sitios visitar?. Es decir, ¿Dónde comienza a buscar el Robot?

Los Robots en general emplean diferentes técnicas de rastreo de direc-

ciones. Lo más normal es comenzar con URL's de Servidores que mantienen muchos enlaces (sitios triviales serían los índices jerárquicos ya creados, como Yahoo!) o con los Servidores de Internet más populares. También podemos comenzar con nuestra propia lista de "*bookmarks*" o favoritos. Lo que siempre será necesario definir será la opción de añadir manualmente un nuevo URL a partir del cual comenzar el análisis.

Hoy día, y para facilitar el trabajo a los Motores de Búsqueda, casi todos los buscadores incorporan unos Formularios para que cualquiera envíe sus propias URL's manualmente (por ejemplo, para dar de alta nuestra página Web personal). El buscador almacenará nuestra petición, la pondrá en cola, y cuando llegue el turno, recibiremos en nuestro Servidor la visita del Robot correspondiente. Adicionalmente se han creado servidores como "Submit-it" (Envíalo), http://www.submit-it.com, que se encargan de enviar la petición del alta en los principales buscadores de Internet (se puede visitar la dirección http://submitit.linkexchange.com/subcats.htm, para ver la lista de buscadores en los que "Submit-it" da de alta las páginas). Normalmente, esta facilidad lleva asociada un coste.

¿QUÉ INFORMACIÓN VA A INDEXAR EL ROBOT?

En realidad, ésta es la primera pregunta que debemos formularnos si vamos a crear un Robot. Esto es tanto como preguntarnos: ¿Con qué fin vamos a crear un Robot Web?. Lo primero que debemos tener claro es si queremos Indexar tanto páginas HTML como otro tipo de recursos: ficheros de imagen, ficheros de sonido, vídeos, etc. O si, por ejemplo, sólo queremos Indexar imágenes, para crear un buscador específico de imágenes.

Incluso dentro de esta primera selección podemos ser aún más específicos. Por ejemplo, si lo que queremos es un buscador de imágenes, podemos distinguir entre imágenes de "fondo" (de *background*) o el resto de imágenes, imágenes de tamaño pequeño o grande (en KBytes), imágenes JPEG o imágenes GIF, imágenes que contienen un enlace o que no, etc.

Los Robots asociados a los buscadores añaden básicamente una única función al proceso descrito: guardar algún tipo de información de las páginas que van visitando

Si nuestra finalidad es indexar páginas HTML (sólo o además de otros recursos), deberemos decidir también que información vamos a guardar acerca de las páginas visitadas e indexadas. Las posibles soluciones pueden ser:

1. *Indexar sólo el título de la página* (es decir la etiqueta <TITLE>): esto presenta el problema de que muchas páginas no presentan esta sección.
2. *Indexar la etiqueta META "DESCRIPTION"*, en la que la página presenta un autoresumen.
3. *Indexar el primer párrafo* (o los primeros párrafos) del contenido (es decir de la parte <BODY>).
4. *Analizar toda la página,* e Indexar aquello que nos parezca oportuno (esto es útil para buscadores Especializados, los cuáles buscan una serie de palabras o cadenas de texto en la página HTML).
5. *Cualquier combinación de las anteriores.*

Como tenemos que simplificar, el Robot que vamos a desarrollar sólo indexará páginas HTML y por tanto sólo analizará los enlaces a desde las páginas a otras páginas HTML. La posible extensión o la modificación del Robot para que indexe imágenes, ficheros de sonido u otros recursos, es un trabajo muy parecido a éste y queda como una posible mejora del mismo.

A su vez, el método de indexación que emplearemos para cada página será el siguiente:

1. Como título de la referencia se utilizará el título de la página si existe (es decir si existe la parte <TITLE>).
2. Si no existe la parte <TITLE>, se empleará como título la primera línea de la etiqueta META "DESCRIPTION".
3. Si tampoco existe esta etiqueta META, utilizaremos la primera línea del texto (dentro de la parte <BODY>, a ser posible una etiqueta <H1>).
4. Como resumen de la referencia, nos serviremos del contenido completo de la etiqueta META "DESCRIPTION".
5. En caso de que no exista, emplearemos el primer párrafo de texto (a ser posible que no sea <H1>).

"ESTADO" ASOCIADO A LAS PÁGINAS

El Robot deberá tener previsto que las páginas de la Web cambian constantemente y por tanto, el hecho de haberlas visitado no debe implicar necesariamente que no se deban analizar de nuevo. Por tanto, para cada pági-

na por la que pase el Robot, deberá almacenar en su Base de Datos una referencia que indique si la página ya fue visitada y cuándo o si está siendo analizada en el presente momento. Además, cuando la página haya sido analizada completamente, también deberá quedar marcado este extremo.

Por tanto, nuestro Robot asociará uno de entre los siguientes "estados" posibles a una página HTML:

1. No visitada.
2. Indexada.
3. En análisis: se deberá indicar el último enlace de la página que ha sido analizado.
4. Analizada: se deberá asociar una "Fecha de finalización", un "tiempo mínimo para la actualización" y una "frecuencia de repetición".

Es decir tendremos una lista de páginas HTML para analizar, de las cuales algunas no habrán sido visitadas todavía, otras ya se habrán indexado, otras estarán siendo analizadas en el momento actual (o se habrán quedado a medias) y por último tendremos las páginas que ya hemos indexado y sobre las que hemos analizado todos sus enlaces. A estas últimas les asociaremos una marca de tiempo que informe del momento en que terminamos de visitarla, una "frecuencia de repetición" que nos indique cuándo debemos volver a verificar sus enlaces y un "tiempo mínimo para la actualización". Si el Robot vuelve a pasar por una página ya analizada, comprueba la marca de tiempo y si ha transcurrido un tiempo mayor a ese tiempo mínimo, la destripamos de nuevo. En caso contrario, ignoramos la página.

De esta manera se puede establecer una secuencia de repetición de páginas ya visitadas y asegurarse de tener actualizadas las páginas que deseemos de acuerdo a nuestros propios criterios.

Estos posibles "estados" de análisis de cada página por parte del Robot, se asociarán en el código del ROBOT a unas constantes de una variable enumerada:

```
type
  TEstadoPagina = (epNoVisitada, epIndexada,
      epEnAnalisis, epAnalizada);
```

PETICIONES DE PÁGINAS

El Robot solicitará las páginas HTML mediante los comandos de petición HTTP. Por tanto deberá seguir la sintaxis de este protocolo. Como ya vimos cuando desarrollamos el Servidor Web, existen tres versiones del protocolo HTTP: 0.9, 1.0 y 1.1. Para el desarrollo del Robot nos ceñiremos –en la medida de lo posible- a la norma HTTP/1.1 que incorpora muchas mejoras respecto a las versiones anteriores.

Una de las mejoras más importantes del protocolo HTTP/1.1, especialmente en el desarrollo de un Robot Web, es la definición por defecto de todas las conexiones como persistentes. Esto quiere decir que todas las peticiones que realiza un cliente a un Servidor Web viajan sobre la misma conexión TCP, con el consiguiente ahorro de procesamiento y de memoria, al no necesitar el establecimiento y liberación de conexiones TCP para cada petición HTTP.

Las ventajas de las conexiones persistentes son muchas:

1. Al solicitar una página HTML lanzamos una petición HTTP por la página y tantas peticiones más como recursos gráficos o de sonido tenga referidos dicha página. Esto significa mucho tiempo de procesamiento y mucha memoria desperdiciada si tenemos que abrir y cerrar conexiones con el mismo Servidor para cada recurso. De ahí que el comportamiento por defecto en la nueva especificación sea el PERSISTENTE.
2. Las peticiones y respuestas HTTP viajan sobre la misma conexión TCP, lo cual permite enviar múltiples peticiones sin tener que esperar cada respuesta. De hecho, el comportamiento por defecto de los clientes debe ser el de enviar las peticiones sin esperar por sus respectivas respuestas. El Servidor, por su parte, debe responder las peticiones en el orden en que se reciben. Por tanto, si desarrollamos el cliente con "multithreading" (multi-hilo) aprovecharemos al máximo el tiempo de CPU y la memoria.
3. Como sólo se establece una conexión TCP para muchas peticiones HTTP, se envían menos paquetes TCP de establecimiento y liberación de conexiones, reduciendo por tanto la carga global de la red.

Además el protocolo HTTP ha ampliado el número de Códigos de Estado que devuelve el Servidor Web en sus respuestas. Se pueden observar estos códigos en la Tabla 1, donde aparecen en negrita los Códigos de Estado que no existían en la versión HTTP/1.0.

En los artículos dedicados a la Creación de un Servidor Web, publicados en los Números 45 y 46 de la Revista Sólo Programadores, ya se revisaron los métodos de petición y de respuesta HTTP, aunque entonces centramos más la atención en las Respuestas HTTP, pues eran las que en ese momento interesaban.

Una petición HTTP incluye en la primera línea el método de acceso al recurso, el identificador del recurso y la versión del protocolo. Las siguientes líneas están formadas por una serie de cabeceras a las que puede seguir el cuerpo del mensaje o ir vacío. La sintaxis es la siguiente:

```
Method URI HTTP-Version #13#10
* (General-header | Request-header | Entity-
      header) #13#10
[ Cuerpo del mensaje ]
```

Tabla 1. Códigos de Estado definidos en la especificación HTTP/1.1
(en negrita, los códigos añadidos al protocolo HTTP/1.0).

CÓDIGO DE ESTADO	CADENA ASOCIADA AL CÓDIGO	SIGNIFICADO
1XX		Códigos 1xx: Indican una respuesta provisional del Servidor Constan sólo de una línea de estado y unas cabeceras opcionales
100	**Continue**	Petición inicialmente atendida. El cliente debe continuar con la petición
101	**Switching Protocols**	La petición de cambio de protocolo en la conexión,realizada con la cabecera "Upgrade", ha sido aceptada
200	OK	Correcto
201	Created	Petición atendida, pero no aceptada todavía
202	Accepted	Petición creada y aceptada, pero pendiente de ser procesada
203	**Non-Authoritative Information**	Información no definitiva, obtenida de una copia local o de terceros
204	No Content	La petición ha sido aceptada, pero no hay información nueva
205	**Reset Content**	El agente de usuario debería borrar la vista que originó la petición
206	**Partial Content**	La petición parcial se ha respondido
300	**Multiple Choices**	El recurso solicitado tiene varias representaciones posibles
301	Moved Permanently	El URL pedido ha cambiado de sitio
302	Moved Temporarily	El URL pedido ha cambiado de sitio temporalmente
303	**See Other**	El recurso solicitado se puede encontrar en otra URL
304	Not Modified	Respuesta a una petición condicional a que haya cambiado el recurso
305	**Use Proxy**	El recurso debe accederse mediante el PROXY indicado en "Location"
400	Bad Request	La sintaxis de la petición es incorrecta
401	Unauthorized	El recurso solicitado requiere una contraseña
402	**Payment Required**	Código reservado para uso futuro
403	Forbidden	El recurso no tiene permitido el acceso
404	Not Found	Recurso no encontrado en el Servidor
405	**Method Not Allowed**	Método no permitido para la URI solicitada
406	**Not Acceptable**	Sólo permitido para los métodos indicados en "Accept"
407	**Proxy Authentication Required**	Igual que 401 pero el cliente debe autenticarse en el PROXY
408	**Request Time-out**	La petición del cliente llegó más tarde que el tiempo de espera del servidor
409	**Conflict**	Conflicto de la petición con el estado actual del recurso
410	**Gone**	Recurso no disponible y ninguna referencia al respecto
411	**Length Required**	Petición sin campo "Content-Length" no aceptada
412	**Precondition Failed**	Las condiciones puestas para la recuperación del recurso no se han cumplido
413	**Request Entity Too Large**	Petición demasiado larga (más larga de lo que el Servidor quiere o puede manejar)
414	**Request-URI Too Large**	URI demasiado larga
415	**Unsupported Media Type**	Formato de petición no soportado
500	**Internal Server Error**	Error del propio Servidor
501	Not Implemented	Característica no implementada en el Servidor
502	Bad Gateway	El Servidor, actuando como Proxy, ha recibido una respuesta invalida de un Servidor
503	Service Unavailable	El Servidor está sobrecargado o está temporalmente desactivado (en mantenimiento)
504	**Gateway Time-out**	Time-out del PROXY
505	**HTTP Version not supported**	Versión de HTTP no soportada

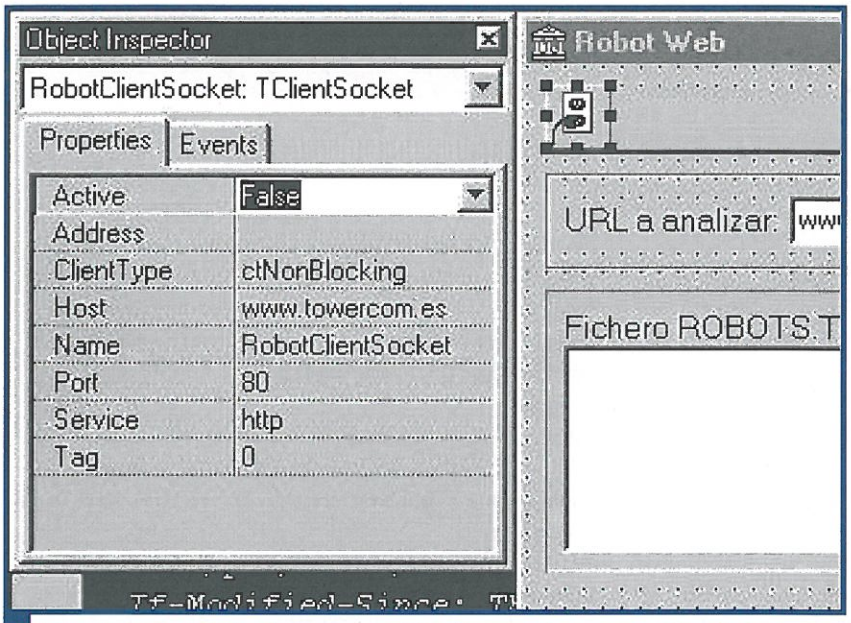


Figura 1. Cuadro de diálogo Properties que deben fijarse en el componente TclientSocket.

La versión del protocolo HTTP, se especifica de la siguiente forma: HTTP/0.9, HTTP/1.0 o HTTP/1.1.

El Método, según la especificación HTTP/1.0, puede ser: GET, HEAD ó POST. La versión HTTP/1.1 define otros métodos adicionales a éstos (OPTIONS, PUT, DELETE y TRACE), pero no los trataremos. El Robot podrá reconocer los métodos aceptados por el Servidor Web al que se conecte a través de la directiva: "Allow" que viaja en la respuesta. Los métodos GET, HEAD y POST deben soportarse en todos los Servidores. En cualquier caso, si un método no es permitido por el Servidor Web, éste debería devolver el código de respuesta *"405 Method Not Allowed"* (405 Método No Permitido), y si no es conocido o no está implementado, debería devolver *"501 Not Implemented"* (501 No Implementado) ver Tabla 1 .

En cuanto a las "Cabeceras", las generales son aquéllas válidas tanto para las peticiones como para las respuestas; las Cabeceras de Petición son específicas para peticiones HTTP y las de entidad definen información sobre el recurso solicitado. El conjunto de Cabeceras definido en la especificación HTTP/1.1 es el siguiente:

general-header = Cache-Control | Connection | Date | Pragma | Transfer-Encoding | Upgrade | Via

request-header = Accept | Accept-Charset | Accept-Encoding | Accept-Language | Authorization | From | Host | If-Modified-Since | If-Match | If-None-Match | If-Range | If-Unmodified-Since | Max-Forwards | Proxy-Authorization | Range | Referer | User-Agent

entity-header = Allow | Content-Base | Content-Encoding | Content-Language | Content-Length | Content-Location | Content-MD5 | Content-Range | Content-Type | Etag | Expires | Last-Modified

No entraremos en el detalle de definir cada una de ellas (pues no es éste nuestro objetivo a conseguir) sino que estudiaremos aquéllas que nos resultarán útiles en el diseño de nuestro Robot.

Una petición HTTP podría tener el siguiente aspecto:

```
GET /articulos/ HTTP/1.1
If-Modified-Since: Thu, 26 Nov 1998 00:01:30
    GMT
Referer: http://www.towercom.es/
Connection: Close
User-Agent: SpRobot/0.01 (Win95; I)
Host: www.towercom.es
Accept: text/html
(línea en blanco)
```

Ésta podría ser la petición HTTP que enviase nuestro Robot Web a la dirección: http://www.towercom.es /articulos/.

En la primera línea aparece el método de petición (*GET*, Obtener), el recurso solicitado (/artículos/) y la versión del protocolo (HTTP/1.1). La cabecera "Host", es la que contiene el Nombre de Dominio del Servidor: www.towercom.es.

Se puede limitar la búsqueda de páginas hasta un límite de saltos desde la primera página o simplemente mediante un temporizador

Se puede observar que se envían algunas de entre todas las posibles cabeceras y que el contenido del mensaje está vacío. Las cabeceras enviadas, aparte de "Host" son:

- *"If-Modified-Since"* (si fue modificado a partir de...) condiciona la respuesta a que el recurso haya sido modificado posteriormente a la fecha que le pasemos como parámetro. Si no ha sido modificado pasada esa fecha, el servidor devolverá una respuesta "304 Not modified" (304 No Modificado).
- *Referer* (Referente) define la URL que nos trajo hasta el recurso solicitado.

- *User-Agent* (Agente de usuario) describe al cliente HTTP. En nuestro caso será el Robot Web.
- *Accept* (Aceptar) determina los recursos a los que se va a limitar a aceptar.
- *Connection:* Permite al cliente especificar opciones que son sólo válidas para esa conexión. Teniendo en cuenta que al ser, por defecto, todas las conexiones en HTTP/1.1 persistentes, si el valor de Connection es *"Close"* (Cerrar), se indica en la petición (o en la respuesta) que la conexión no debe considerarse persistente. El valor *"Close"* es obligatorio para todas las mensajes de aplicaciones HTTP/1.1 que no soporten conexiones persistentes.

PRIMERA PETICIÓN HTTP

De acuerdo a lo que vimos en el artículo del mes pasado, y para cumplir el "protocolo de exclusión de robots" el primer archivo que debe recuperar nuestro Robot de cada nuevo servidor al que se conecte es "ROBOTS.TXT" que deberá ir situado en el directorio raíz. Por tanto para el ejemplo anterior, la primera petición tendría la siguiente forma:

```
GET /robots.txt HTTP/1.1
Referer: http://www.towercom.es/
User-Agent: SpRobot/0.01 (Win95; I)
Host: www.towercom.es
Accept: text/txt
(línea en blanco)
```

Lógicamente, el fichero "ROBOTS .TXT" puede no existir en el Servidor, lo cual en un principio nos da pleno acceso a todas sus páginas públicas.

En función del contenido del fichero "robots.txt" nos bajaremos o no las páginas HTML, dependiendo de su situación en el Servidor y los permisos de acceso que nos hayan sido concedido. Posteriormente, una vez bajada cada página HTML, analizaremos sus etiquetas META (si existen) y en función de ellas la indexaremos e inspeccionaremos sus enlaces.

COMPONENTE DELPHI

Lógicamente el componente de la VCL que vamos a utilizar será el "TclientSocket", y sobre él desarrollaremos todos los métodos de petición del protocolo HTTP.

Lo primero que analizará el Robot será el fichero "ROBOTS.TXT" y para cada página HTML mirará las etiquetas META

Las propiedades que hay que establecer en este componente, se muestran en la Figura 1. En realidad, la única imprescindible es, o bien la propiedad *"port"* (puerto TCP) o bien la propiedad *"service"* (servicio). Cualquiera de estas dos determina la otra, si bien es mejor utilizar *"service"* porque es más fácil confundirse en la asignación del número de puerto de un servicio.

La propiedad *"Host"* (Servidor) y la propiedad *"Address"* (Dirección IP) también indican una misma cosa: el Servidor al que vamos a conectarnos. Y al igual que antes, siempre será más difícil confundirse al escribir el nombre de dominio de un Servidor que al escribir su dirección IP.

Por último, señalar que además de los eventos "OnRead" y "OnWrite" (que ya utilizamos en el TserverSocket, en nuestro Servidor Web), este componente incorpora otros que pueden resultar muy útiles: "OnConnect", "OnConnection", "OnDisconnect" y "OnLookup".

La secuencia de disparo de estos eventos es la siguiente:

1. Un Socket cliente abre una conexión.
2. Se dispara el evento "OnLookup", antes de que se intente localizar el Socket Servidor.
3. El Socket se inicializa para la notificación de eventos.
4. Se dispara un evento "OnConnecting" cuando el Socket Servidor ha sido localizado.
5. El Socket servidor acepta la petición de conexión y se completa por el Socket cliente.
6. Cuando la conexión se establece, se dispara un evento "OnConnect".

CONCLUSIÓN

Hemos dado los primeros pasos para crear un Robot Web, analizando primero qué es exactamente lo que nos proponíamos. Luego se han descrito algunas de las características más importantes de la versión 1.1 del protocolo HTTP. Posteriormente hemos entrado en materia, generando las primeras líneas de código de nuestro futuro Robot Web.

En el siguiente artículo de la serie terminaremos el Robot que hemos comenzado y comenzaremos a crear la estructura de la Base de Datos que almacenará las páginas indexadas.

Si alguien desea enviar algún comentario sobre este artículo (o anteriores), puede hacerlo enviando un E-mail a la dirección de la revista:

solop@towercom.es
o a la dirección del autor:
elastra@redestb.es

Windows Scripting Host (II)

Adolfo Aladro (aaladro@arrakis.es)

En el artículo anterior presentamos Windows Scripting Host y vimos algunos ejemplos sencillos que utilizaban los nuevos objetos que esta tecnología pone a nuestra disposición para acceder a todo tipo de elementos del sistema. En el presente artículo vamos a abundar en todos esos conceptos.

INTRODUCCIÓN

Aunque son muchas y de muy diversa índole las tareas que podemos efectuar con *WSH*, probablemente el acceso al sistema de ficheros será una de las que más utilizaremos. Por ello vamos a empezar viendo cómo podemos acceder y manejar el sistema de archivos del sistema.

El objeto FileSystemObject (FSO) proporciona una forma de manejar directorios y archivos siguiendo la filosofía de orientación a objetos

El objeto *FileSystemObject (FSO)* proporciona una forma de manejar directorios y archivos siguiendo la filosofía de orientación a objetos, es decir, utilizando las propiedades y métodos. Además la tabla 1 contiene los objetos y colecciones del modelo de objetos que se deriva del *FSO*. Para realizar los programas que utilizan el modelo de objetos propuesto por el *FileSystemObject* podemos seguir una serie de pasos que a continuación os pasamos a detallar.

PASO 1. CREAR UNA INSTANCIA DEL OBJETO FILESYSTEMOBJECT

Antes de nada debemos crear una instancia del objeto *FSO*. El listado 1 nos va a mostrar la mejor manera de realizarlo tanto en *VBScript* como en *JScript*.

Listado 1. Creación de una instancia del objeto FSO.

```
VBS   Dim mi_fso;
Set mi_fso =
      CreateObject("Scripting.FileSystemObject");
JS    var mi_fso;
mi_fso = new
      ActiveXObject("Scripting.FileSystemObject");
```

PASO 2. UTILIZAR EL MÉTODO QUE SEA APROPIADO

Una vez que tenemos creada una instancia del objeto *FSO* debemos utilizar el método que sea preciso en cada caso. Podemos crear nuevos objetos utilizando los métodos *CreateFolder* o *CreateTextFile* (El objeto *FSO* no admite la creación de unidades). Podemos borrar objetos utilizando los métodos *DeleteFile* y *DeleteFolder*, o podemos copiar o mover archivos o directorios.

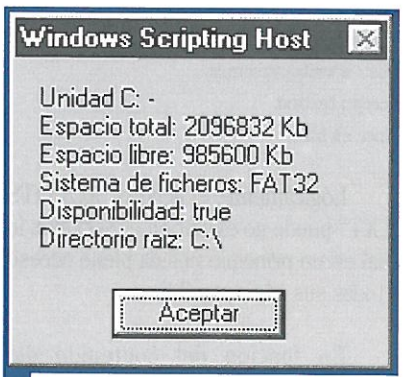


Figura 1. Resultado de ejecutar el script del listado 3.

Algunas de las funcionalidades que proporciona el objeto *FSO* a través de sus métodos son redundantes. Por ejemplo, podemos copiar un archivo utilizando el método *CopyFile* del objeto *FSO* o podemos utilizar el método *Copy* del objeto *File*.

PASO 3. ACCEDIENDO A LAS PROPIEDADES DEL OBJETO

Una vez que tenemos un manejador mediante el cual podemos acceder al objeto, estaremos en condiciones de o leer sus propiedades. Por ejemplo, el listado 2 muestra el código necesario para obtener el nombre de un determinado directorio.

Listado 2. Acceso a las propiedades de un directorio.

```
VBS  Set fldr = fso.GetFolder("c:\tmp")
Wscript.Echo "El nombre del directorio es: " &
     fldr.Name
JS   var fldr = fso.GetFolder("c:\\tmp");
Wscript.Echo("El nombre del directorio es: " +
     fldr.Name);
```

Una vez que tenemos creada una instancia del objeto FSO debemos utilizar el método que sea preciso en cada caso

Listado 3. Script que muestra la información acerca de una unidad.

```
WScript.Echo(MostrarInfoUnidad("c:"));

Function MostrarInfoUnidad(nombre_unidad) {
 var fso, unidad, respuesta ="";
 fso = new
     ActiveXObject("Scripting.FileSystemObje
     ct");
 unidad =
     fso.GetDrive(fso.GetDriveName(nom-
     bre_unidad));
 respuesta += "Unidad " +
     nombre_unidad.toUpperCase() + " - ";
 respuesta += unidad.VolumeName + "\n";
 respuesta += "Espacio total: " +
     unidad.TotalSize / 1024;
 respuesta += " Kb" + "\n";
 respuesta += "Espacio libre: " +
     unidad.FreeSpace / 1024;
 respuesta += " Kb" + "\n";
 respuesta += "Sistema de ficheros: " + uni-
     dad.FileSystem;
 respuesta += "\n";
 respuesta += "Disponibilidad: " +
     unidad.isReady;
 respuesta += "\n";
 respuesta += "Directorio raiz: " +
     unidad.RootFolder;
 respuesta += "\n";
 return respuesta;
```

Tabla 1. Objetos y colecciones de objetos.

Objeto	Descripción
Drives	Colección de objetos que proporciona una lista de todos las de unidades (discos duros, CD-ROM, discos RAM, etc.) de ese sis tema (lógicas o físicas). Aquellas unidades que sean removibles no tienen que estar presentes obligatoriamente para aparecer en esta lista
Drive	Objeto gracias al cual podemos obtener información de una deter minada unidad.
Folders	Colección que proporciona una lista de todos los directorios que "cuelgan" de un determinado directorio.
Folder	Objeto gracias al cual podemos obtener información de un deter minado directorio.
Files	Colección que proporciona una lista de todos los archivos que están en un determinado directorio.
File	Objeto gracias al cual podemos obtener información de un deter minado archivo.
FileSystemObject	Objeto principal del grupo gracias al cual podemos acceder a directorios, archivos y unidades.
TextStream	Objeto gracias al cual podemos leer y escribir archivos de texto.

TRABAJANDO CON UNIDADES

Gracias al objeto *FSO* podemos conocer todo tipo de información de las

Tabla 2. Propiedades de las unidades.

Propiedad	Significado
TotalSize	Tamaño total en bytes.
AvailableSpace	
FreeSpace	Tamaño disponible en bytes.
DriveLetter	Letra asignada.
DriveType	Tipo de unidad (Unidad removible, fija, de red, CD-ROM o disco RAM).
SerialNumber	Número de serie.
FileSystem	Tipo de sistema de archivos (FAT, FAT32, NTFS, etc.).
IsReady	Si está o no disponible.
ShareName	
VolumeName	Nombre compartido o del volumen.
Path	
RootFolder	Ruta del directorio raíz

Tabla 3. Métodos involucrados en el manejo de directorios.

Método	Significado
FileSystemObject.CreateFolder	Crear.
Folder.Delete	
FileSystemObject.DeleteFolder	Borrar.
Folder.Move	
FileSystemObject.MoveFolder	Mover.
Folder.Copy	
FileSystemObject.CopyFolder	Copiar.
Folder.Name	Nombre.
FileSystemObject.FolderExists	Comprueba si existe.
FileSystemObject.GetFolder	Instancia del directorio.
FileSystemObject.GetParentFolderName	Nombre del directorio padre.
FileSystemObject.GetSpecialFolder	Ruta del directorio del sistema.

distintas unidades del sistema. La tabla 2 muestra las propiedades a las que tenemos acceso a través de un objeto de unidad y el listado 3 muestra cómo podemos acceder a dichas propiedades.

TRABAJANDO CON DIRECTORIOS

Del mismo modo, trabajar con directorios es una tarea muy sencilla. La Tabla 3 muestra los métodos relacionados con el manejo de directorios. El listado 4 muestra un ejemplo de cómo utilizar dichos métodos.

TRABAJANDO CON ARCHIVOS

Todas las operaciones que podemos realizar con archivos se engloban dentro de dos categorías: la primera es aquella donde se encuentran todas las acciones encaminadas hacia la manipulación del contenido mismo de los archivos; la segunda es aquella en donde se engloban todas las operaciones de borrado, copiado o movimiento de archivos. A continuación vamos a ir viendo ejemplo por ejemplo el tratamiento de los archivos desde las dos vertientes.

Para leer un archivo de texto utilizaremos los métodos Read, ReadLine, y ReadAll del objeto TextStream

Listado 4. Script para acceder a los directorios.

```
WScript.Echo(MostrarInfoDir());
function MostrarInfoDir() {
 var fso, fldr, respuesta = "";
 fso = new ActiveXObject("Scripting.FileSystemObject");
 fldr = fso.GetFolder("c:");
 respuesta += "El nombre de directorio por defecto es: " + fldr + "\n";
 respuesta += "Se encuentra la unidad " + fldr.Drive + "\n";
 if (fldr.IsRootFolder) {
  respuesta += "Es el directorio raíz.";
 } else {
  respuesta += "No es el directorio raíz.";
 }
 respuesta += "\n\n";
 fso.CreateFolder("C:\\tonto");
 respuesta += "Hemos creado el directorio C:\\tonto" + "\n";
 fso.DeleteFolder ("C:\\tonto");
 respuesta += "Hemos borrado el directorio C:\\tonto" + "\n";
 return respuesta;
}
```

Creación de archivos de texto

Existen tres formas de crear un archivo de texto dependiendo de si utilizamos el método *CreateTextFile*, el método *CreateTextFile* o el método *OpenTextFile*. Las tres formas son equivalentes y realmente no hay demasiada diferencia entre unas y otras. El listado 5 muestra estas tres formas.

Listado 5. Formas de crear un archivo.

```
1    var fso, f1;
fso = new
      ActiveXObject("Scripting.FileSystemObjec
      t");
f1 = fso.CreateTextFile("c:\\prueba.txt", true);
2    var fso, ts;
var ForWriting= 2;
fso = new
ActiveXObject("Scripting.FileSystemObject");
ts = fso.OpenTextFile("c:\\ prueba.txt",
      ForWriting, true);
3    var fso, f1, ts;
var ForWriting = 2;
fso = new
      ActiveXObject("Scripting.FileSystemObje
      ct");
fso.CreateTextFile ("c:\\ prueba.txt");
f1 = fso.GetFile("c:\\prueba.txt");
ts = f1.OpenAsTextStream(ForWriting, true);
```

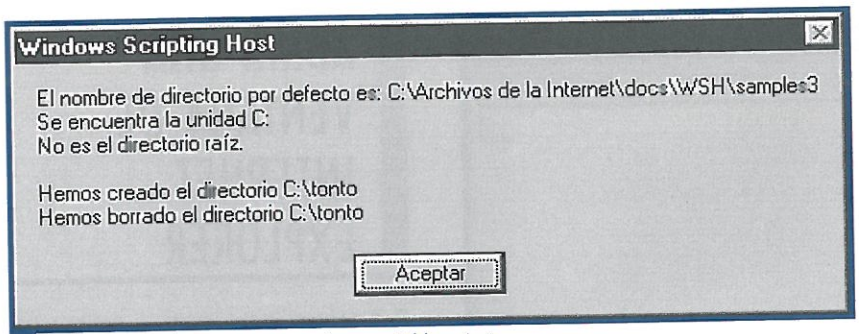

Windows Scripting Host
El nombre de directorio por defecto es: C:\Archivos de la Internet\docs\WSH\samples3
Se encuentra la unidad C:
No es el directorio raíz.
Hemos creado el directorio C:\tonto
Hemos borrado el directorio C:\tonto
Aceptar

Figura 2. Resultado de ejecutar el script del listado 3.

- **Escribir archivos de texto**

Podemos escribir en los archivos de texto siguiendo los siguientes pasos: primero abrimos el archivo, después escribimos en él y finalmente lo cerramos.

Para abrir un archivo podemos utilizar el método *OpenAsTextStream* del objeto *File* o bien el método *OpenTextFile* del objeto *FileSystem Object*.

Para escribir en el archivo el objeto *TextStream* proporciona tres métodos: *Write*, *WriteLine* y *WriteBlank Lines*. El primero de ellos escribe los datos sin añadir un carácter salto de línea al final de los mismos. El segundo, por el contrario, añade el carácter fin de línea. Por último, el último método escribe uno o varios caracteres en blanco.

Todas las operaciones que podemos realizar con archivos se engloban dentro de dos categorías

Para cerrar un fichero utilizaremos el método *Close* bien del objeto *TextStream* o del objeto *FileSystem Object*.

El listado 6 muestra un *script* de ejemplo en el que creamos un archivo y escribimos alguna información en él. El resultado de la ejecución del *script* se puede apreciar en la figura 3.

Listado 6. Script que crea un archivo de texto.

```
CrearArchivo();

function CrearArchivo() {
 var fso, tf;
 fso = new
      ActiveXObject("Scripting.FileSystemObje
      ct");
 tf = fso.CreateTextFile("c:\\tmp\\prueba.txt",
      true);
 tf.WriteLine("Probando la escritura...") ;
 tf.WriteBlankLines(3) ;
 tf.Write("Esto es una prueba.");
 tf.Close();
}
```

Del mismo modo, trabajar con directorios es una tarea muy sencilla

- **Leer ficheros**

Para leer un archivo de texto utilizaremos los métodos *Read*, *ReadLine*, y *ReadAll* del objeto *TextStream*. El primero de ellos sirve para leer un determinado número de caracteres del archivo. Utilizaremos el segundo en el caso de que queramos ir leyendo línea a línea.

Listado 7. Script que crea un archivo de texto.

```
WScript.Echo(LeerFichero());
function LeerFichero(){
 var fso, f1, ts, respuesta="";
 var ForReading = 1;
 fso = new ActiveXObject("Scripting.FileSystemObject");
 ts = fso.OpenTextFile("c:\\tmp\\prueba.txt", ForReading);
 while (!ts.AtEndOfStream)
   respuesta += ts.ReadLine( );
 ts.Close();
 return respuesta
}
```

Listado 8. Script que abre una ventana de Internet Explorer.

```
var ie = new ActiveXObject("InternetExplorer.Application");
ie.left=50;
ie.top=50;
ie.height=300;
ie.width=400;
ie.toolbar=false;
ie.StatusBar=false;
var loc = WScript.ScriptFullName;
var dir = loc.substring(0,1 + loc.lastIndexOf("\\")) + "interfaz.htm";
ie.navigate(dir);
ie.visible=1;
```

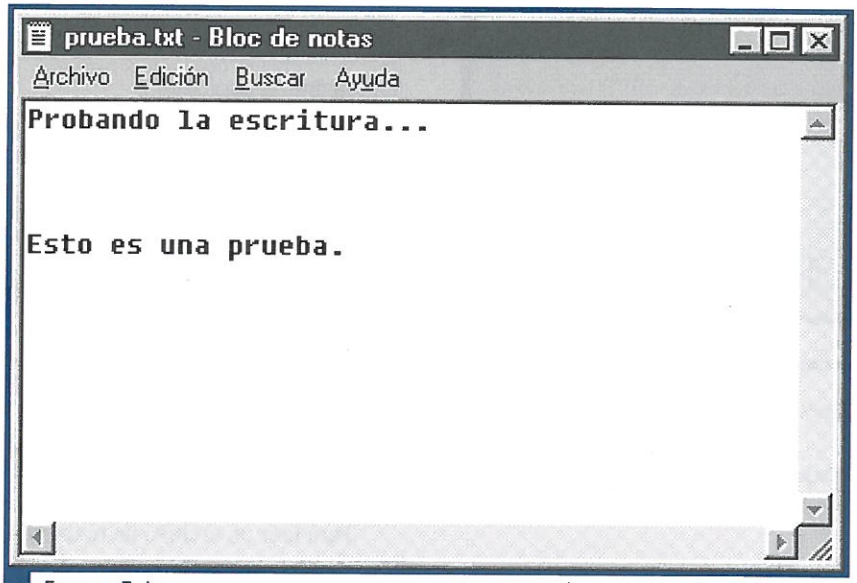


Figura 3. Fichero que se crea como consecuencia de ejecutar el programa del listado 6.

Finalmente el último se utilizará para leer todo el archivo de una vez. En el caso de que utilicemos los dos primeros métodos es posible obviar un determinado intervalo de datos gracias a los métodos *Skip* y *SkipLine*. El listado 7 muestra un ejemplo de cómo podemos leer ficheros.

Abordaremos temas muy diversos dentro de la programación WSH

El objeto *TextStream* dispone de dos propiedades las cuales sirven para crear bucles de lectura tal y como podemos ver en el ejemplo. La propiedad *AtEndOfLine* será **true** si el puntero de lectura del fichero se sitúa inmediatamente antes del carácter fin de la línea. Además la propiedad *AtEndOfStream* informa acerca de si hemos llegado al final del archivo.

- **Copiando, moviendo y borrando ficheros**

Como es fácil deducir, estas tres operaciones se pueden realizar de una manera muy sencilla, simplemente utilizando los métodos expuestos en la tabla 3.

Bien, hasta aquí hemos podido ver la mayoría de las posibilidades que ofrece *WSH* en lo que al sistema de archivos respecta. A partir de este punto vamos a adentrarnos en otros ejemplos más complejos gracias a los que abordaremos temas muy diversos dentro de lo que engloba la programación *WSH*. De esta manera, veremos el mayor número de ejemplos posibles que abarquen todo tipo de cuestiones distintas.

ABRIR UNA VENTANA DE INTERNET EXPLORER

En primer lugar vamos a realizar un *script* que abra una determinada página *HTML* en el navegador *Internet Explorer*. El código correspondiente podremos analizarlo en el listado 8, observando que no resulta nada complicado. Se trata tan sólo de crear una instancia del objeto *InternetExplorer.Application* y acceder a algunas de sus propiedades y métodos.

Para abrir un archivo podemos utilizar el método OpenAsTextStream

Tal y como se puede apreciar en el listado abrimos una ventana del navegador y hacemos que ésta vaya a la página "interfaz.htm" (Página cuya

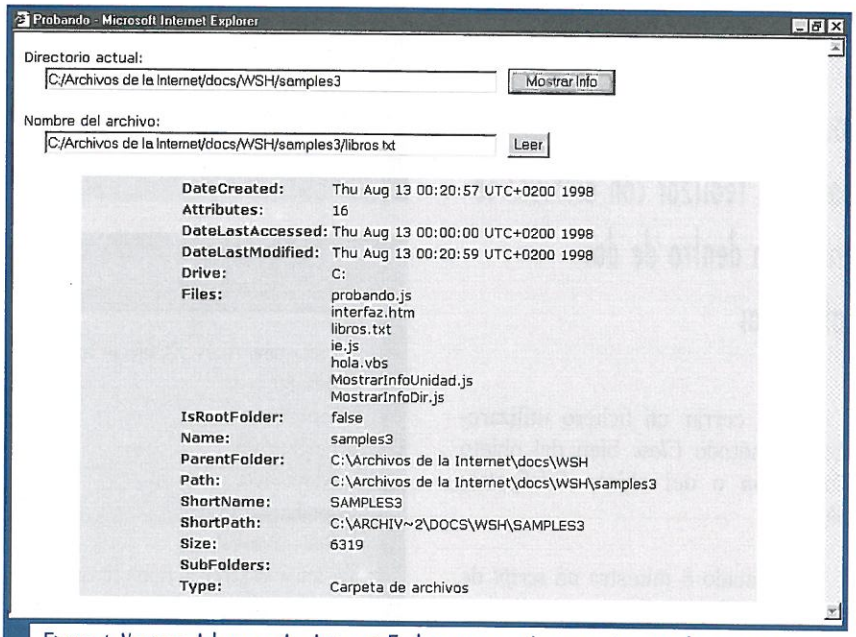


Figura 4. Ventana del navegador Internet Explorer que se abre tras ejecutar el script que aparece en el listado 8.

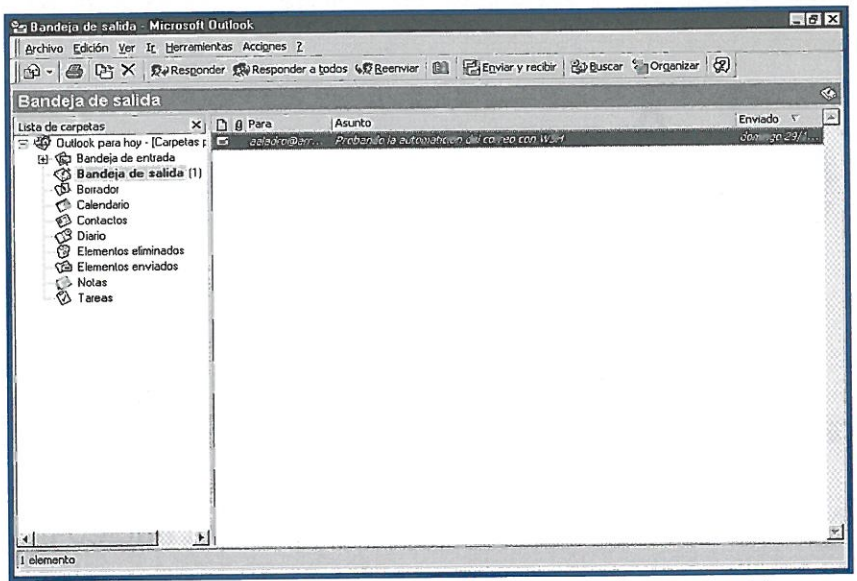


Figura 5. Bandeja de salida de la aplicación Outlook después de ejecutar el script del listado 9.

dirección *URL* completa calculamos a partir del directorio donde se encuentra el propio *script*). La figura 4 muestra la ventana del navegador que se abre con los resultados que se obtienen después de hacer clic en el botón **Mostrar Info**. Como ya se puede deducir, este botón hace que se muestre la información relativa al directorio que se encuentra especificado en el campo de texto correspondiente, y que por defecto toma el valor del directorio donde reside el *script* que abre la ventana del navegador. El botón **Leer** sirve para mostrar los contenido del archivo cuya ruta absoluta se encuentra en la casilla de texto correspondiente.

El botón Leer sirve para mostrar los contenidos del archivo cuya ruta absoluta se encuentra en la casilla de texto correspondiente

Los *scripts* que ejecutan las funciones asignadas a los botones se hayan dentro de la misma página. De esta manera acabamos de integrar el entorno *Web* (que en definitiva es el padre de la tecnología *WSH*) y los *scripts* del sistema.

MANDAR UN MENSAJE DE CORREO ELECTRÓNICO CON OUTLOOK

Siguiendo con esta tan interesante línea de integración de los *scripts* con el resto de los elementos del sistema vamos a ver cómo podemos mandar un mensaje de correo electrónico a través de la aplicación Microsoft *Outlook*, pero utilizando un *script*. El listado 9 nos muestra de manera clara el código de este *script*. De nuevo simplemente creamos una instancia del objeto correspondiente a la aplicación y llamamos a sus métodos y propiedades. La figura 5 muestra la bandeja de salida de la aplicación *Outlook* después de ejecutar el *script*.

El objeto TextStream dispone de dos propiedades las cuales sirven para crear bucles de lectura

IMPRIMIR UN DOCUMENTO WORD

Finalmente vamos a pasar a ver un ejemplo en el que imprimimos un documento cualquiera Word desde un *script*. A continuación en el listado 10 podremos aprender y averiguar el código del *script* que necesitaríamos para llevar a cabo esto.

Listado 9. Script que manda un mensaje de correo electrónico a través de la aplicación Outlook.

```
var outlook = new ActiveXObject("Outlook.Application");
var mi_correo = outlook.CreateItem(0);
mi_correo.Recipients.Add("aaladro@arrakis.es");
mi_correo.Subject = "Probando la automatición del correo con WSH";
mi_correo.Body = "Probando...";
mi_correo.Send();
```

Listado 10. Script que imprime un documento Word.

```
var word = new ActiveXObject("Word.Application");
word.Documents.Open("C:\\Mis documentos\\artículos\\wsh(II)\\wsh(II).doc");
word.Options.PrintBackground = false;
word.ActiveDocument.PrintOut();
```

Visual Basic 6.0

Jordi Agost (agost@eup.udl.es)

En este artículo presentamos las principales novedades de Visual Basic 6.0, de reciente aparición en nuestro país. Se han hecho modificaciones en el nucleo del lenguaje, en Internet, se han mejorado los asistentes e incluye novedades en los controles.

INTRODUCCIÓN

Su lenguaje se basa en el *Basic* de antaño, aunque debido a sucesivas evoluciones poco tiene que ver con él. Es necesario especificar que *Visual Basic* se vende independientemente o bien como parte del conjunto de herramientas de programación *Visual Studio 6.0*. Asimismo también cabe destacar que en el momento de escribir este artículo, Microsoft ha puesto a disposición del usuario en su *Web* el *Service Pack 1* de *Visual Studio 6.0* el cual corrige algunos errores.

VERSIONES DE VISUAL BASIC 6.0

Como ya viene siendo tradicional, el lenguaje *Visual Basic* se ha presentado en diferentes versiones para que cada uno pueda elegir la que más se adapta a sus necesidades y a sus gustos. En *Visual Basic 6.0* encontramos tres versiones o ediciones diferentes del mismo lenguaje:

EDICIÓN DE APRENDIZAJE DE VISUAL BASIC 6.0

El equipo de desarrollo de *Microsoft* supone que el usuario de esta versión no tiene ninguna experiencia previa en programación. Básicamente está pensada para estudiantes, aficionados o gente que requiera una cierta idea sobre como desarrollar aplicaciones para Windows.

EDICIÓN PROFESIONAL DE VISUAL BASIC 6.0

Esta versión esta diseñada para profesionales individuales o programadores de una determinada corporación que necesiten crear una aplicación disponible para *Internet* o bien una determinada aplicación cliente-servidor.

EDICIÓN EMPRESARIAL DE VISUAL BASIC 6.0

Diseñada para equipos de programadores, la versión empresarial de Visual Basic 6.0 ,que tienen que crear aplicaciones distribuidas cliente-servidor y necesiten un alto rendimiento. O bien para aquellos equipos que necesitan crear aplicaciones de *Internet* o *intranet*.

NOVEDADES EN EL LENGUAJE

Vamos a ver y a conocer todas las novedades que ha sufrido el núcleo principal del lenguaje Visual Basic. Para una mejor lectura y una mejor comprensión vamos a proceder a dividirlos en dos subgrupos: el primero de ellos será el de las novedades del lenguaje en sí mismas y el segundo las novedades en lo referente a funciones de cadena.

NOVEDADES EN EL LENGUAJE

Dichas modificaciones o ampliaciones corresponden al núcleo del lenguaje y por consiguiente van a ser aplicables a todas las ediciones de *Visual Basic 6.0*. Una de las novedades es que ahora las funciones y los diferentes procedimientos de propiedades van a ser capaces de devolver *arrays*. Además dentro del campo de los *arrays* vamos a poder utilizar uno dinámico en la izquierda de una determinada asignación (aunque dicha operación continuará vetada a los vectores o *arrays* de un tamaño fijo). Otra limitación del lenguaje que se ha eliminado es que los tipos definidos por el usuario pueden ser los argumentos o los tipos de retorno de propiedades públicas y de métodos públicos. En todas las ediciones se ofrece unos objetos de sistema llamados *File System Object u* Objetos del Sistema de Ficheros (*FSO* u *OSF* abreviadamente). Dicho modelo *FSO* posee los siguientes objetos:

En todas las ediciones se ofrecen unos objetos de sistema llamados File System Object

- **Drive:** objeto que permite obtener más información sobre las distintas unidades de disco que posee el sistema, información tal como, por ejemplo, su nombre de recurso compartido o el espacio. No es necesario que estén conectadas al sistema las unidades de las cuales vamos a obtener información, es decir, puede, por ejemplo accederse a ellas desde una red de área local. Además las unidades pueden ser también de *CD-ROM*, discos *RAM*, etc.
- **Folder:** dicho objeto permite crear, borrar y mover carpetas, así como requerir al sistema información diversa sobre los directorios.
- **Files:** objeto que permitirá crear, borrar o mover ficheros y preguntar al sistema todo tipo de información sobre los ficheros.
- **FileSystemObject:** el principal objeto de todo el grupo y es el que nos proporciona acceso al sistema de ficheros del ordenador.
- **TextStream:** objeto que permitirá leer y escribir ficheros de texto.

La función *CallByName* permitirá acceder a una determinada propiedad o método especificando una cadena que contenga el nombre de la propiedad o el método. Por ejemplo para invocar el método *Move* de una caja de texto ahora podemos utilizar la siguiente sintaxis:

```
CallByName Text1, "Move", vbMethod, 100,
    100
```

Otra novedad disponible en todas las versiones es que ahora con la función *CreateObject*, la cual nos permitía crear una referencia a un objeto *ActiveX*, podremos especificar entre sus argumentos si lo deseamos, un nombre de máquina para así crear objetos en máquinas remotas.

La función *StrConv* ha sido convenientemente ampliada y ahora tiene un argumento opcional:

```
StrConv(string, conversion, LCID)
```

Dicho argumento permite especificar un identificador de localización distinto al identificador establecido por defecto que en este caso sería el del sistema. Se ha añadido un nuevo objeto llamado *Dictionary* (Diccionario) que viene a suplir alguna de las limitaciones del objeto *Collection*. Es más fácil de utilizar que éste, más flexible y además genera menos errores de ejecución .

NUEVAS FUNCIONES DE CADENA

Han sido añadidas un total de catorce nuevas funciones, en todas las versiones del lenguaje, para ampliar su funcionalidad. Veamos a continuación un resumen de dichas funciones:

- *WeekDayName*: esta función devolverá una cadena indicando un determinado día de la semana.
- *StrReverse*: retornará una cadena en la cual el orden de los caracteres ha sido invertido.
- *Split*: retornará un *array* unidimensional que contendrá un determinado número de subcadenas.
- *Round*: devolverá un número redondeado.
- *Replace:* devuelve una cadena en la que se ha reemplazado una determinada subcadena por otra subcadena un determinado número de veces.
- *MonthName* : esta función retornará una cadena indicando un mes determinado del año.
- *Join:* devuelve una cadena formada de la unión de un número de subcadenas contenidas en un *array*.
- *FormatPercent:* devuelve una expresión convenientemente formateada como un porcentaje, es decir, multiplicada por cien con el carácter de porcentaje (%).
- *FormatNumber:* devolverá una expresión formateada como un número.
- *FormatDateTime:* devolverá una expresión convenientemente formateada como una fecha o bien como una hora.
- *FormatCurrency:* esta función retornará una expresión formateada como un valor de moneda, usando para ello el símbolo de moneda que esté definido en el panel de control de nuestro sistema.
- *Filter Function:* retornará un *array* conteniendo un subconjunto de un *array* de cadenas. Dicho subconjunto estará basado en un determinado criterio de filtro.

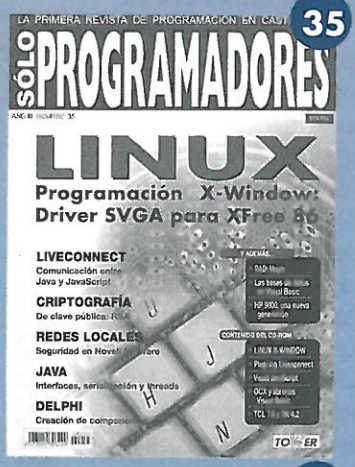
ASISTENTES

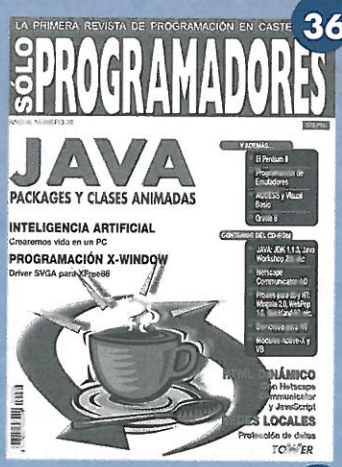
Vamos a ver en este apartado varias novedades de los asistentes. Por una parte, en esta nueva versión del lengua-

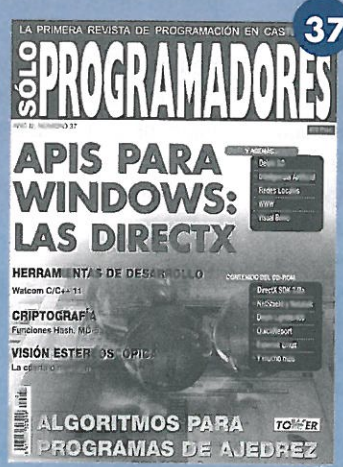
números atrasados

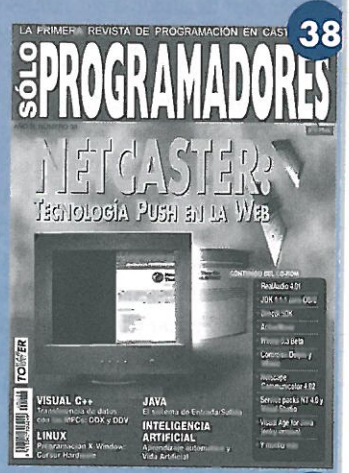
SÓLO PROGRAMADORES

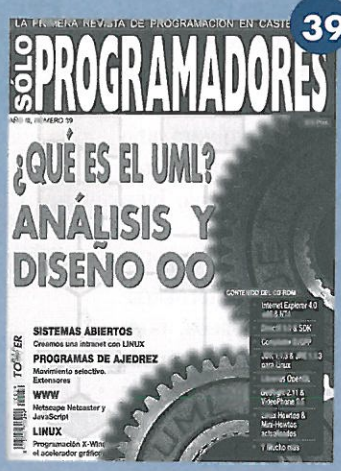
completa ya tu colección











SÓLO PROGRAMADORES

☑ **Sí, deseo suscribirme a la revista SÓLO PROGRAMADORES durante un año (12 números) eligiendo la siguiente modalidad.** (Marcar con una X)

☐ Suscripción **NORMAL** 9.350 ptas.

☐ Suscripción **ESTUDIANTES** 7.050 ptas.

Nombre .....Apellidos .....

Domicilio: calle .....Nº .....Piso.....

C.P: .....Población: .....Provincia: .....

Teléfono: .....Fax:.....E-mail: .....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº..... .... Fecha de caducidad..... / .....

☐ Domiciliación bancaria. (Rellenar código cuenta cliente)
☐ Contra-reembolso del importe más gastos de envío.
☐ Cheque a nombre de TOWER COMMUNICATIONS, S.R.L. que adjunto.
☐ Giro Postal (adjunto fotocopia del resguardo).

CODIGO DE CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

•ESTA OFERTA ANULA LAS ANTERIORES
•Oferta válida hasta fin de existencias •Válido para Peninsula y Baleares

---

SÓLO PROGRAMADORES

☑ **Sí, deseo recibir los siguientes números atrasados .....**
.....
..... Importe total ..... ptas.

de SÓLO PROGRAMADORES al precio unitario de 975 ptas. (IVA incluido)

números 1, 2, 3, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 23, 24 y 26 agotados

Nombre .....Apellidos .....

Domicilio: calle .....Nº .....Piso.....

C.P: .....Población: .....Provincia: .....

Teléfono: .....Fax:.....E-mail: .....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº..... .... Fecha de caducidad..... / .....

☐ Contra-reembolso del importe más gastos de envío.
☐ Cheque a nombre de TOWER COMMUNICATIONS, S.R.L. que adjunto.
☐ Giro Postal (adjunto fotocopia del resguardo).

ESTA OFERTA ANULA LAS ANTERIORES
Oferta válida hasta fin de existencias •Válido para Peninsula y Baleares

PEGAR AQUÍ

Rte.: _____

_____

_____

HOJA DE PEDIDO

RESPUESTA COMERCIAL
Autorización nº 14.107
B.O. de C. nº 36
del 18.04.97

NO
NECESITA
SELLO
(A franquear
en destino)

TOWER
COMMUNICATIONS
APARTADO FD Nº 214
28080 MADRID

suscríbete

a Sólo Programadores y consigue un magnífico descuento

suscripción normal

ahorro **20%**

12 revistas (1 año) por sólo...

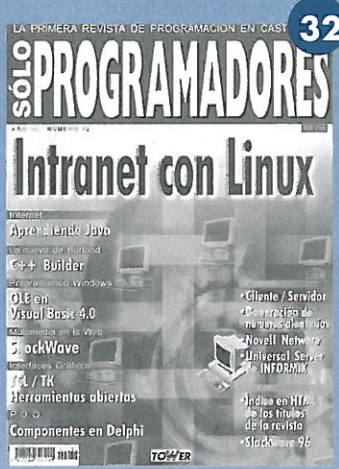
9.350 ptas.

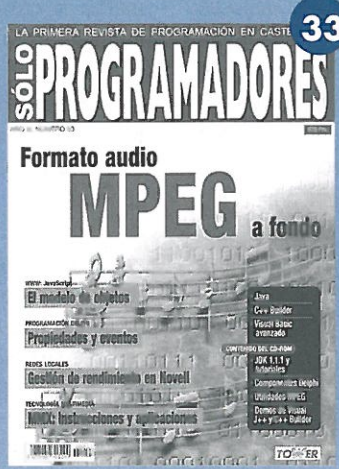
suscripción estudiantes (carreras técnicas)

ahorro **40%**

12 revistas (1 año) por sólo...

7.050 ptas.





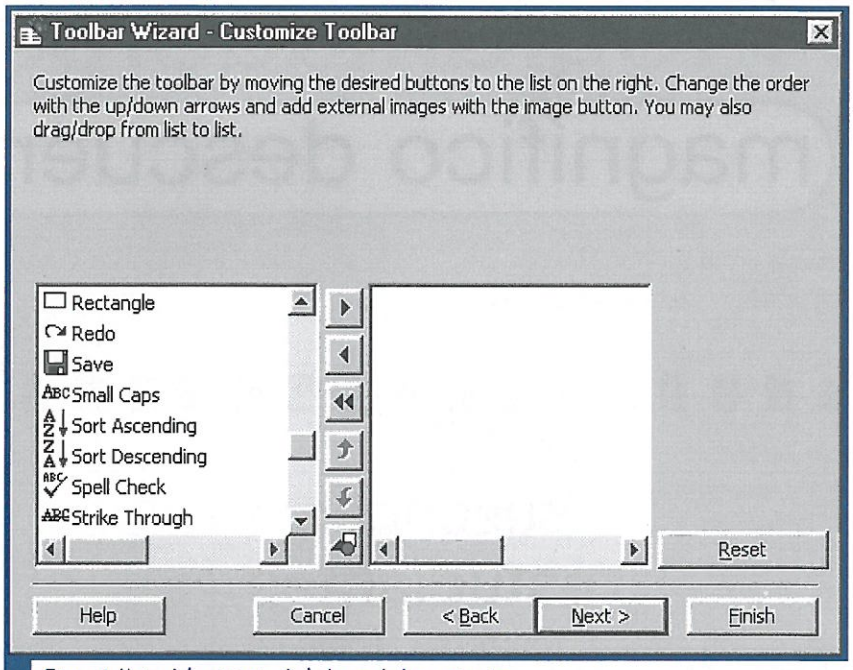

Toolbar Wizard - Customize Toolbar
Customize the toolbar by moving the desired buttons to the list on the right. Change the order with the up/down arrows and add external images with the image button. You may also drag/drop from list to list.
Rectangle
Redo
Save
Small Caps
Sort Ascending
Sort Descending
Spell Check
Strike Through
Reset
Help
Cancel
< Back
Next >
Finish

Figura 1. Vista del asistente de la barra de herramientas.

je se han incluido nuevos asistentes y se han mejorado algunos de los que ya existían. Así como también se ha mejorado su control de carga. Vamos a ver a continuación las novedades en este campo:

Algunos de los asistentes han sufrido diversas mejoras, y entre todos quizás el que mayor remodelación ha sufrido es el instalador de aplicaciones, también conocido como *Setup Wizard*. Podremos ejecutarlo como un *Add-In* desde dentro de *Visual Basic* o bien desde la línea de órdenes como una línea más de un hipotético proceso *bat*. Dicho asistente se ha remodelado en todas las versiones del lenguaje y además ahora los ficheros creados por dicho programa (los famosos ficheros *cab*) pueden desplegarse hacia un directorio del disco, tal y como se hacía hasta ahora, pero también hacia un servidor de *Web* o un recurso de red compartido. Las nuevas tecnologías que incorpora *Visual Basic*, también han sido contempladas en dicho asistente. De este modo dicho asistente soporta ahora las tecnologías siguientes: *ADO*, *OLE DB*, *RDO*, *ODBC* y *DAO*.

Y también podremos realizar instalables para los nuevos tipos de aplicación existentes: aplicaciones del tipo *DHTML* (*Dynamic HTML o HTML* dinámico) y aplicaciones del tipo *IIS* (*Internet Information Server* o Servidor de Información de *Internet*). Por otra parte también se ha mejorado el control sobre el menú de inicio y sobre los grupos e iconos que crea la instalación de nuestro programa.

El *Asistente para Objetos de Datos* se encuentra presente en la edición profesional y empresarial de Visual Basic, y permitirá crear clases y controles de usuario, los cuales van a estar unidos a un determinado entorno de datos. En cierta manera podríamos decir que automatiza la creación de objetos para una arquitectura de tres capas.

Este asistente se ha ampliado y nos permite ahora construir formularios a partir de tablas o consultas almacenadas, utilizando para ello el control de datos *ADO*. Dicho asistente se encuentra a su vez integrado dentro de otros asistentes tales como el creador de aplicaciones, el de gráficos y el de la rejilla *FlexGrid*. Se encuentra disponible en las ediciones profesional y empresarial.

También modificado en todas las ediciones del lenguaje, nos permite ahora grabar nuestras preferencias como perfiles, permitiendo de este modo crear múltiples aplicaciones con un mismo formato. Asimismo desde dentro de este asistente podremos lanzar el asistente de *Formulario de datos* y el *Asistente de la Barra de Herramientas*. Además como toque final diremos que ahora los menús son completamente personalizables.

El *Asistente de la barra de herramientas* tal y como hemos dicho este asistente puede lanzarse desde dentro del asistente de aplicaciones o bien individualmente desde el menú de *Add-Ins*. Se encuentra disponible en todas las ediciones y permite crear una barra de herramientas personalizada.

El *Asistente o utilidad generadora de clases* ahora soporta más funcionalidades. Por ejemplo en la lista de argumentos se soporta ahora las siguientes palabras clave: *Optional*, *ByVal*, *ParamArray*. También soporta el uso de Enumeraciones y el valor por defecto en la lista de argumentos. Disponible en las versiones profesional y empresarial.

El *Diseñador de Add-Ins* nos permitirá empezar de forma inmediata una aplicación, diseñando los complementos por defecto, el nombre de la aplicación, su versión y otras características.

NOVEDADES RELATIVAS A INTERNET

Con la versión 5.0 del lenguaje *Visual Basic* se estrenó en el mundo de *Internet*. Ahora con sus nuevas características intenta afianzarse dentro de dicho mundo. Vamos a hacer un breve repaso de las más significativas.

Podremos crear aplicaciones *IIS* (*Internet Information Server*), es decir aplicaciones que usen el código de *Visual Basic* para responder a las demandas de un usuario desde un navegador.

Las *clases de Web* son componentes de tipo *COM* cuya utilización se basa en un servidor *Web*, que ofrecen a los programadores de *Visual Basic* la posibilidad de escribir código para "personalizar" respuestas de dirección *URL* desde dentro del entorno de desarrollo de *Visual Basic*. La aplicación resultante es una aplicación multiplataforma basada en servidor y accesible desde cualquier explorador *HTML* en cualquier plataforma. La idea de crear las *WebClasses* surgió para que los programadores pudiesen crear una aplicación para el usuario basada en el *Web* para aplicaciones existentes de *Visual Basic*, o para generar aplicaciones basadas en *Web* mediante el modelo controlado por eventos de *Visual Basic* para separar la parte lógica de la aplicación de la interfaz del usuario.

El *HTML dinámico* dentro de *Microsoft Internet Explorer* es una tecnología que expone las propiedades, métodos y eventos para cada elemento de una página *Web*. Una aplicación de *HTML* dinámico es una aplicación que realizaremos en *Visual Basic* y la cual va a utilizar una combinación de *HTML* dinámico y código de *Visual Basic*, en una aplicación interactiva basada en el navegador. Dicha aplicación residirá en la máquina del navegador, en donde interpretará y responderá a las acciones que haga el usuario final en el navegador. En su forma más simple una aplicación *DHTML* puede ser una simple página *HTML*, que use el código de *Visual Basic* junto con el modelo *DHTML* para responder a ciertas acciones que pueden ocurrir en la página. Dichas acciones pueden venir por parte del usuario (como por ejemplo movimientos del ratón o clics del mismo) o acciones por parte del navegador, como por ejemplo la carga de un gráfico o la apertura de una determinada página. Dentro de un nivel más elevado podríamos hacer tareas más complejas como recuperar datos de una página y usarlos para consultar una base de datos, o bien cambiar el aspecto visual y comportamiento de una página o crear elementos *HTML* para insertarlos dentro de una página.

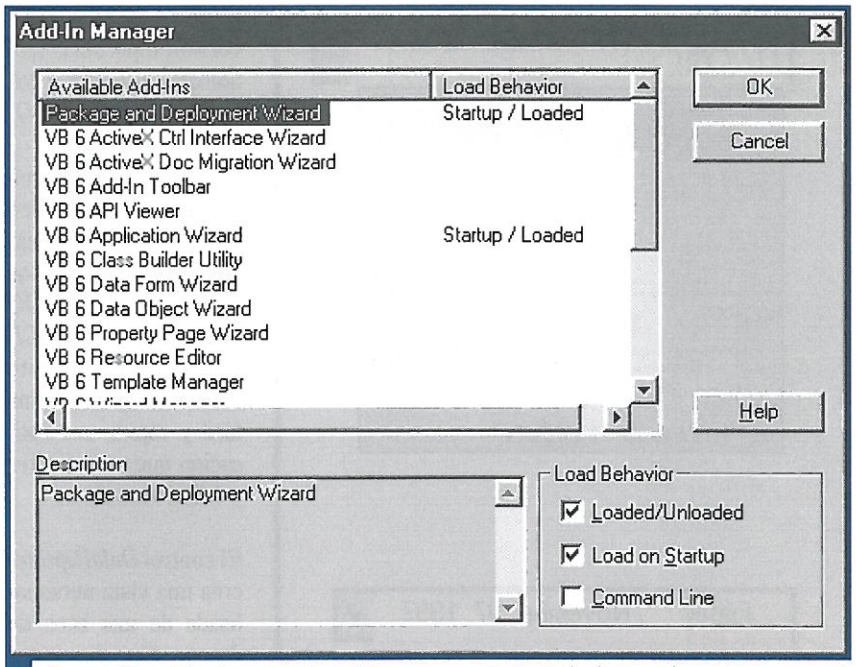


Figura 2. Vista del asistente de la Add-Ins, nos permite controlar los complementos.

Tal y como hemos dicho en el apartado de asistentes, también podemos utilizar el *Instalador de aplicaciones* para descargar nuestra aplicación hacia un sitio *Web*.

Las *Operaciones Asíncronas* han sido ampliamente remodeladas (en todas las versiones del lenguaje) con el fin para facilitarnos más información sobre la operación que está en marcha en ese momento. Así podemos utilizar algunas propiedades y algunos métodos nuevos como por ejemplo el evento *AsyncReadProgress*, las propiedades *ByteMax*, *BytesRead*, *Status*, *Status Code* y *Target*. Además el método *AsyncRead* y el evento *AsyncReadComplete* han sido ampliados.

Otra novedad se encuentra en el soporte de *Internet Explorer 4.x* para la descarga de documentos *ActiveX*. El proceso de descarga de documentos *ActiveX* en el navegador de *Microsoft Internet Explorer 4.x* funciona ahora de la misma forma que la descarga de controles *ActiveX*. Como consecuencia de este hecho no necesitamos una pagina separada de *HTML* para lanzar nuestro objeto a descargar. También el navegador puede descargar de una forma automática la última versión del documento *ActiveX* si éste no se encuentra instalado en la máquina del usuario. Además se dota a los usuarios de la posibilidad de ver el documento *ActiveX* dentro del navegador.

NOVEDADES EN CONTROLES

Quizás las características más impactantes ya sea por su vistosidad o por su utilidad, o las primeras en que nos fijamos sean los nuevos controles. Hemos dividido este apartado en dos subapartados, uno que hace referencia a los nuevos controles y otro que nos habla de nuevas características en los controles ya existentes.

Figura 3. Vista del control DateTimePicker.

NUEVOS CONTROLES

Dentro de los nuevos controles nos encontramos con algunos de ellos que son una revisión de otros antiguos u otros totalmente nuevos. Veámoslos a continuación:

El *Control de datos ADO* presente en todas las ediciones es un nuevo control que sigue la tecnología *OLE DB*. Funciona como el control intrínseco de datos y el control de datos remoto y nos permite crear una aplicación de bases de datos con un mínimo de código.

El control *CoolBar* presente en las ediciones Profesional y Empresarial es un control contenedor que puede utilizarse para crear barras de herramientas configurables por el usuario. Las barras resultantes son parecidas a las incorporadas por el *Microsoft Internet Explorer*.

También en todas las versiones se han añadido controles de lista de datos, y lista desplegable de datos (control *DataList* y *DataCombo*) que son versiones de los controles *DBList* y *DBCombo* existentes anteriormente. Dichos controles en esta nueva versión soportan la tecnología *OLE DB* y los nuevos controles *ADO*.

Lo mismo sucede con el control *DataGrid* (en todas las versiones) que ahora soporta los nuevos controles *ADO* y la tecnología *OLE DB*. Y nos permite construir de una forma fácil y rápida una aplicación que permita ver y editar *recordsets*.

El control *DataRepeater* crea una vista personalizada de una base de datos. Su funcionamiento es similar a los formularios de *Access*, y puede contener cajas de texto, de validación u otros controles.

El control *DateTimePicker* hace aparecer un calendario que se despliega para que así podamos introducir rápidamente fechas y horas.

Similar a este control nos encontramos con el control *MonthView* que permite al usuario tanto al elegir fechas como al elegir rangos de fechas de un calendario gráfico. Es aplicable a las ediciones Profesional y Empresarial.

También en estas versiones encontramos el control de barras de *scroll* planas (**Flat Scrollbar**). Dicho control ofrece la misma funcionalidad que el control de barras estándar de *Windows*, pero con diferencias en la interfaz visual. Dicho control puede aparecer visualmente como el estándar tridimensional de facto "barra de desplazamiento", como una barra plana (dos dimensiones) o como una barra de desplazamiento plana con flechas que aparecen biseladas cuando el mouse se pone sobre ellas. Es visualmente consistente con *Microsoft Internet Explorer 4.0*.

En todas las ediciones se ha incluido además una versión actualizada del control *FlexGrid*. Sumado a todas las características de dicho control encontramos que además puede mostrar una jerarquía de objetos *recordset*, de la tecnología *ADO* (*ActiveX Data Object*). Cada *recordset* se mostrará con una banda separadora en la cuadricula y además puede ser formateado independientemente de los otros.

El control *ImageCombo* está presente en todas las ediciones de *Visual Basic* y además se pueden añadir imágenes a la lista de elementos, para hacer así una lista *combo box* mucho más amigable.

CARACTERÍSTICAS NUEVAS EN LOS CONTROLES

También algunos de los controles existentes han ampliado su funcionalidad de alguna u otra forma. Veamos una breve descripción de alguno de estos cambios, los cuales son aplicables a todas las ediciones de *Visual Basic 6.0*.

En los controles que usan la validación de datos y el evento *Validate* se ha añadido una propiedad llamada *CausesValidation*. Dicha nos permite validar el contenido del control antes de que éste pierda el foco. Además mediante la programación podemos añadir o borrar controles de un formulario o de formulario a formulario.

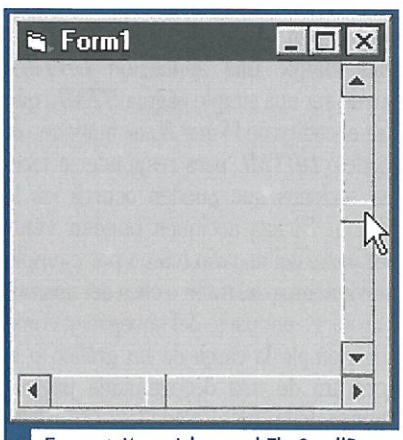


Figura 4. Vista del control FlatScrollBar.

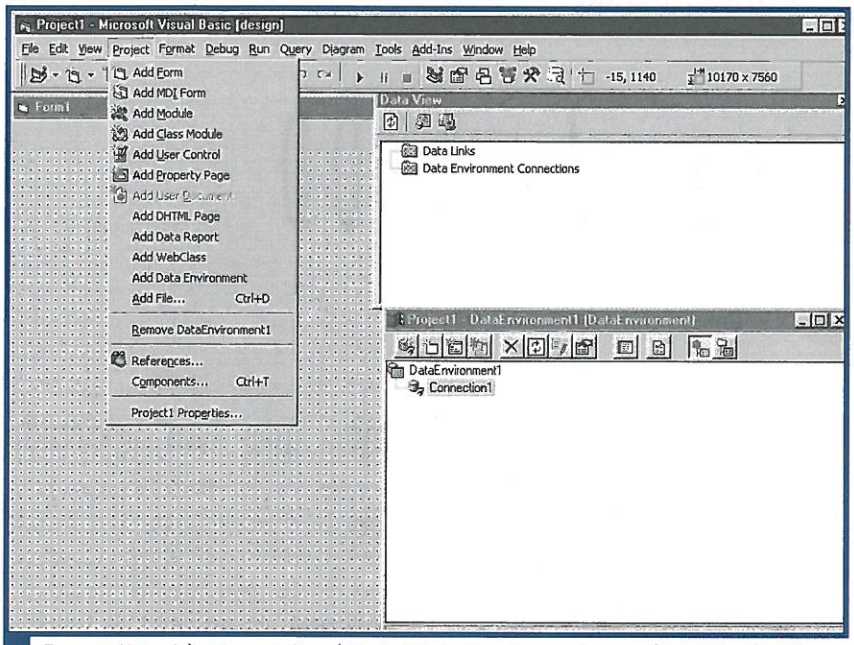


Figura 5. Vista del entorno de Visual Basic 6.0, donde podemos observar el menú desplegado.

También la función *LoadPicture* ha sido modificada, ya que ahora siempre que hagamos referencia a ella podremos especificar el tamaño de la imagen que cargar, así como su profundidad de color. También se ha añadido el método *ValidateControl*, el cual valida el contenido del último control que existe en el formulario antes de que salgamos del mismo.

ACCESO A DATOS

Dentro de este apartado encontramos novedades que ya han sido nombradas en el apartado de las novedades de asistentes o de las novedades de los controles, por lo que ya no las vamos a repetir, limitándonos en este apartado a las novedades tecnológicas.

La tecnologia *ADO* o *ActiveX data Objects* presupone un modelo más simple de objetos para acceder a datos, así como una mayor integración con otras tecnologías propietarias y no propietarias de Microsoft. Además ofrece una interfaz común tanto para acceder a datos locales como remotos, también *recordsets* jerárquicos, etc. Presente en todas las ediciones.

También en todas ellas se ha dado soporte a la tecnología *OLE DB*. Se trata de un conjunto de interfaces que siguen la arquitectura *COM* (*Common Object Model*) y que hacen posible que las aplicaciones tengan un acceso uniforme a los datos guardados en diferentes fuentes, ya sean relacionales o nó. Dichas interfaces soportan la funcionalidad de *DBMS* (*DataBase Manager System*) del sistema que accedemos, haciendo posible que este comparta los datos. La tecnología *ADO* será la forma a través de la cual los programadores accederán a la tecnología *OLE DB*. Todos los controles que acceden a datos, tal y como se ha podido ver en la sección de controles, son capaces de aprovechar toda esta tecnología.

El *Diseñador de entorno de datos*, presente en la edición Profesional y Empresarial, ofrece una forma interactiva de crear objetos *ADO* (*ActiveX Data Objects*) en tiempo de diseño. Soporta también la creación de agregados, jerarquías, agrupamientos, arrastrar y soltar, etc.

En la edición Empresarial hay dos herramientas: el *Diseñador de consultas* y el *Diseñador de bases de datos* que visualmente permiten crear y modificar esquemas de bases de datos, así como consultas.

Los objetos *Data Report*, de la ediciones Profesionales y Empresariales, premiten utilizar el arrastrar y soltar para así crear informes de cualquier *recordset*, incluyendo también los de tipo jerárquico. También en las mismas ediciones podemos crear *Controles de usuario* y clases que pueden ser fuentes de datos hacia las cuales podemos ligar otros controles.

Otra novedad es la ventana de *Visualización de datos (Data View Window)* utilizada para mostar todas las bases de datos a las que estamos conectados, sus tablas, vistas, procedimientos guardados en ellas, etc.

El *Editor de SQL*, presente en la edición Empresarial únicamente, permite añadir nuevos procedimientos para bases de datos tipo *SQL Server* u *Oracle*.

También en todas las ediciones podemos pasar *recordsets* de tipo *ADO* entre diferentes procesos y entre diferentes máquinas (via *HTTP* o via *DCOM*), lo cual se convierte en una magnifica forma de mover datos entre las diferentes capas de una aplicación multicapa.

Los *Objetos de formato*, presentes en todas estas aplicaciones, permiten una conversión en ambos sentidos entre una base de datos y un control unido a ella. Cuando el valor es leído de una base de datos, entonces el objeto *Format* añade el consiguiente formato al control ligado a dicha información.

Hasta aquí hemos podido ver un pequeño resumen de todas las características que se incluyen en esta nueva versión del lenguaje.

Nuevas técnicas de BBDD

Juan Manuel Menéndez (jfrias@ibm.net)

A lo largo de este artículo conoceremos las soluciones que han aportado importantes empresas para los sistemas de bases de datos.

SISTEMAS DE BASES DE DATOS RELACIONALES

Los sistemas relacionales de bases de datos forman (*SGBDR*), junto con la lógica de los negocios y las interfaces gráficas los tres pilares de los sistemas de información actuales, pero de estos tres elementos ninguno quizá como los SGBDR tienen más claras sus fronteras.

Veremos cómo empresas proveedoras de *SGBDR* tales como *Informix*, *PostGres*, *Oracle* y *Microsoft* han aportado sus soluciones a estos temas.

BASES DE DATOS EN ENTORNOS CLIENTES/SERVIDOR

Modo 1. *Presentación distribuida*: Datos, Lógica y parte de la presentación son gestionadas por el servidor, parte de la presentación por el cliente, la arquitectura 3270 es de este modelo.

Modo 2. *Cliente gráfico*: Datos y lógica residen en el servidor, los clientes sólo tienen capacidad de presentación, un ejemplo de esta es la arquitectura *Web* en sus inicios.

Modo 3. *Aplicación distribuida*: Datos en los servidores y lógica repartida entre el servidor y el cliente, la interfaz de usuario íntegramente en el cliente.

Modo 4. *Servidor de Datos*: Los datos residen en los servidores y la lógica en los clientes junto con la presentación de usuario. La mayor parte de las aplicaciones pertenecen a este modelo, como es el caso de *Cartera*, que si bien el sistema completo es un sistema distribuido, las aplicaciones que lo componen en su mayoría funcionan en este modo.

Modo 5. *Datos distribuidos*: En este modelo los datos están distribuidos entre el servidor y el cliente.

BASES DE DATOS EN LA RED

El último avance de la tecnología ha acercado al usuario (empresarial o doméstico), un potencial de información prácticamente ilimitado, y para manejar esta información cada vez se necesitan sistemas de gestión de aplicaciones más complejos. *CGI* es incapaz de dar respuestas a las nuevas necesidades de procesos y los sistemas de gestión de bases de datos deben ser capaces de responder al nuevo reto. Una base de datos en la red es utilizada de varias formas. Para comprender cómo funciona la integración vemos los diversos tipos de páginas *HTML* que pueden estar contenidas en un sitio *Web*. Las páginas siguen generalmente uno de estos tres modelos:

- Páginas estáticas que entregan generalmente contenido *HTML*.
- Páginas dinámicas que proporcionan la capacidad de entregar contenido dinámico al usuario. Normalmente esta información está basada en una que el usuario ha entregado con anterioridad. Estas páginas son generadas usando información procedente de una base de datos a la que se accede en modo lectura.
- Páginas activas que proporcionan la capacidad de soportar una lógica, y gestionan transacciones contra bases de datos a las que pueden acceder incluso en modo escritura. Las aplicaciones que tiene estas

páginas suelen necesitar elementos adicionales que realizan la correcta gestión de todas las particularidades que existen en este tipo de aplicaciones.

Las bases de datos, como se ha podido observar tienen un papel importante en las páginas dinámicas y las páginas activas.

Listado 1. Esquema de un procedimiento PL/SQL.

```
PROCEDURE demo
IS
   Var1              NUMBER:
   I            BINARY INTEGER;
   J            BINARY INTEGER := 1;
BEGIN
   << Lógica del procedimiento >>
   CURSOR  c_var1 is
   SELECT campo1 from tabla 1;
OPEN c_var1
FETCH c_var  INTO var1
END
```

ORGANIZACIÓN FÍSICA

Una de las tareas de los sistemas de gestión de bases de datos relacionales, consiste en presentar a los programadores una visión lógica según un modelo aceptado independientemente de la organización física de los mismos. Esta independencia ha conseguido que la arquitectura física de los datos sea la más optima para el sistema sobre el que va trabajar la base datos. Vamos a ver como un fabricante de *SGBDR*, *Informix*, tiene tres distintos tipos de organización física para sus bases de datos.

Informix Online puede gestionar el espacio que tiene asignado para las bases de datos de dos maneras:

- Dispositivos *cooked*.
- Dispositivos *raw*.

En los primeros, sobre un *filesystem standard* del sistema se crea un fichero con el tamaño deseado y se asigna al sistema de gestión de la base de datos, que crea las estructuras necesarias para el mantenimiento de la base de datos, pero desde el punto de vista del sistema operativo lo que se observa es un fichero. En los segundos todo el espacio del *filesystem* es gestionado directamente por *Informix Online*, mediante rutinas de acceso de bajo nivel. Indistintamente del tipo de dispositivo creado el espacio físico asignado para las bases de datos se organiza en cuatro unidades diferentes:

- *Chunks*.
- Extensiones (*Extents*).
- Paginas.
- Páginas de *Blob*.

Sobre estas unidades de espacio físico se organizan cinco unidades lógicas asociadas con la gestión de la base de datos:

- *Blobspace*.
- *Dbspace*.
- *Database*.
- *Tblspace*.
- Tablas.

El *chunk* es la mayor unidad de espacio físico que *Informix Online* dedica para el almacenamiento de datos. Si el *chunk* pertenece a una asignación de espacio *cooked*, cada *chunk* está representado por el nombre completo de un fichero. Si por el contrario pertenece a una asignación *raw*, el *chunk* está representado por el fichero correspondiente al dispositivo en el directorio */dev*.

Todo el espacio dentro de un *chunk* aparece divido en páginas, las cuales son la unidad de entrada/salida para el gestor, y al estar tan ligado a la estructura física, una vez definido para un *chunk* no puede ser cambiado.

Los *dbspaces* contienen las tablas y las bases de datos y están formados por uno o más *chunks*, y si es necesario, según aumente el tamaño de las bases de datos, es posible añadir nuevos *chunks*. Los *blobs* (*Binary Large ObjectS*) son las estructuras que tiene *Informix* para almacenar grandes campos de texto como pueden ser documentos, imágenes, etc. Por sus características tienen una forma de gestión diferenciada de los datos normales. Se almacenan en *blobspaces*, y su unidad de transferencia entrada/salida es la página *blob*, que debe ser múltiplo de la página de datos.

Un poco más arriba se ha comentado que los *dbspaces* contienen tablas y bases de datos. El lector familiarizado con las bases de datos se habrá extrañado ante esta afirmación, después de todo las bases de datos están formadas por tablas luego la expresión parece un poco redundante. Es cierto en parte pero en la arquitectura que estamos describiendo se llama *database* al catálogo de tablas del sistema que definen dicha base de datos. Esta separación del espacio de la *database* y las tablas, aunque parezca complicada, permite una gran flexibilidad a la hora de administrar la base de datos, pues permite tener repartidas las diversas tablas que contienen los datos en diferentes *chunks*.

Los *tblspace* se refiere a todo el espacio asignado a tablas, y están forma-

ACCESO A BASES DE DATOS DESDE VISUAL BASIC

Una vez que se entra en el mundo de *Microsoft* es obligatorio hacer una revisión general al sistema de acceso a datos creado por este fabricante. Para complicar la cosa ha creado varios modelos de objetos para el acceso a datos, entre los que destacan:

- *DAO. Direct Data Objects*. Permite el acceso al motor *Jet* de *Access*.
- *RDO. Remote Data Objects*. Acceso a bases de datos por medio de *ODBC*.
- *OLE DB*. Acceso a datos mejorando ciertas limitaciones de *ODBC*.
- *ADO. ActiveX Data Objects*. Encapsula *OLE DB* con *ActiveX*.

El uso de estos objetos es similar, siendo necesario definir los objetos, instanciar cada uno de ellos y comenzar a operar y disparar losprocedimientos de evento. En el listado 4 se puede ver un esquema general de lo expuesto anteriormente.

EL CONTROL DE DATOS

Una de la piedras angulares de la programación con bases de datos desde *Visual Basic* es el Control de Datos (*Data Control*). Las propiedades y métodos de este control dan al programador las herramientas necesarias para el acceso a datos, como se puede ver en la tabla 1.

Los métodos de los *RecordSet* no pertenecen al control de datos sino al objeto *RecordSet* que a su vez pertenece al control. Por eso en la referencia al método se hace a través de objeto de acceso a datos de la forma:

```
NombreControlDatos.Recordset.Metodo
```

Los *RecordSet* pueden ser de diversos tipos: *dynaset*, *snaphost* o tabla. Explicar la diferencias entre estos tipos sale del ámbito de un curso básico de programación, utilizando *RecordSet* de tipo *dynaset* se puede realizar cualquier acción sobre la base de datos. Además del acceso a los datos por medio de *RecordSet*, el control de datos permite el acceso por medio de los controles enlazados.

A modo de práctica se va explicar como se accede por medio de los controles de datos a una base de datos *Access*. Con este fin se incluye en el *CD-ROM* un programa que gestiona los artículos de revistas. Este programa es un formulario *MDI* con tres formularios simples en su interior a los que se puede acceder a través de la opción del menú **View**. A partir de ahora todas las referencias vienen hechas sobre elementos de este programa, y todo se hará más comprensible si el lector lo ejecuta antes.

CONTROLES ENLAZADOS A DATOS

Los controles enlazados permiten por medio de su propiedad *DataSource* enlazar con un control de datos y mediante la propiedad *DataFied* con la lista de campos que suministra el control de datos previamente enlazado. Una vez establecida esta doble asociación dentro la caja de texto la propiedad *Text* de la misma se enlaza con el valor que existe en ese campo para el registro actual que tenga cargado el control de datos. Ya creados los formularios por medio el asistente de aplicaciones se introducen datos en las diversas tablas. Para introducir datos con los formularios creados con el asistente primero hay que pulsar el botón **Add**. Al pulsar este botón las cajas se quedan en blanco y una vez escrita la información en las cajas de texto se pulsa sobre **Add** para seguir introduciendo más registros o si es el último se pulsa el botón **Refresh**.

El primer paso consiste en cargar las tablas auxiliares con los nombres de nuestras revistas favoritas y en la de *Temas*, aquellos sobre los que queremos guardar una relación. La siguiente tarea consiste en realizar el mismo trabajo para los artículos, pero aquí la tarea es un poco más tediosa ya que el número de campos a introducir es considerablemente mayor. Se va a utilizar la información existente en otras tablas para introducir datos en la tabla de *Artículos*.

Los controles *DBList* y *DBCombo* son una caja de lista y una caja combinada enlazadas a un control de Datos y que

Tabla 1. Métodos más comunes del objeto RecordSet.

Método	
AddNew	Añade un registro a la tabla
Close	Cierra el objeto RecordSet
Delete	Elimina el registro activo de la tabla
FindFirst	Busca el primer registro que cumpla unos criterios
FindLast	Busca el último registro que cumpla unos criterios
FindNext	Busca el siguiente registro que cumpla unos criterios
FindPrevious	Busca el anterior registro que cumpla unos criterios
GetRows	Recupera múltiples filas
Move	Mueve la posición del registro (según parámetros)
MoveFirst	Va al primer registro
MoveLast	Avanza al último registro
MoveNext	Avanza un registro
MovePrevious	Retrocede un registro
OpenRecordset	Abre el objeto RecordSet
Requery	Actualiza los datos
Update	Actualiza un registro.

automáticamente reflejan el contenido del *RecordSet* en la lista. Antes de incluirlos en el formulario se hará alguna modificación en las cajas de texto para adecuarlas al tamaño de los campos. Después se eliminan las cajas de texto de los campos *Temas* y *Revistas*. El siguiente paso es añadir dos controles de datos, con las mismas propiedades que el control de datos existente en el formulario excepto *RecordSource* que tendrá los valores de la tabla *Revistas* para *Data2* y *Temas* para *Data3*. Como se accederá en consulta, para evitar que sean modificadas por error se establecerá la propiedad *ReadOnly* de ambas con el valor **True**. Por último se pondrá la propiedad *Visible* a **False** para que estos controles no sean visibles en el formulario. Se llevan dos componentes de tipo *DBCombo* (o bien *DBList* pues el funcionamiento es igual) y se da valor a sus propiedades de enlace de datos. Estas propiedades permiten trabajar con dos controles: uno como origen de los datos y otro como destino de los mismos. Las relacionadas con el destino de datos son:

- *DataField*: Nombre del campo de destino.
- *DataSource*: Nombre del control de datos de destino.

Las propiedades relacionadas con el origen de los datos son:

- *RowSource*: Nombre del control de datos de origen.
- *ListField*: Nombre del campo que aparece en la lista.
- *BoundColumn*: Nombre del campo seleccionado.

Aclarar los conceptos *ListField* y *BoundColumn*. Se amplía la tabla de *Revistas* y el nombre de la revista se pone el *ISBN* (*International Standard Book Number*). En la tabla de *Artículos* el campo *Revista* va a contener el *ISBN* en vez del nombre de revista. En la lista del control aparece el nombre de la revista, pero el campo seleccionado puede ser el *ISBN*. Las propiedades serán:

Listado 4. Programa en Visual Basic que muestra el acceso a datos.

```
Sub RecordsetX()

   Dim dbsNeptuno As Database
   Dim rstTable As Recordset
   Dim rstDynaset As Recordset
   Dim rstSnapshot As Recordset
   Dim rstForwardOnly As Recordset
   Dim rstBucle As Recordset
   Dim prpBucle As Property
   Set dbsNeptuno = OpenDatabase("Neptuno.mdb")
   With dbsNeptuno
      ' Abre cada tipo de objeto Recordset.
      Set rstTable = .OpenRecordset("Categorías",dbOpenTable)
      Set rstDynaset = .OpenRecordset("Empleados", dbOpenDynaset)
      Set rstSnapshot = .OpenRecordset("Compañías de envíos", dbOpenSnapshot)
      Set rstForwardOnly = .OpenRecordset ("Empleados", dbOpenForwardOnly)
      Debug.Print "Recordsets en la colección " &" Recordsets de dbsNeptuno"
      ' Enumera la colección Recordsets.
      For Each rstBucle In .Recordsets
         With rstBucle
            Debug.Print "    " & .Name
         ' Enumera la colección Properties de cada
          'objeto Recordset. Bloquea cualquier
            ' propiedad cuyo valor no es válido
            ' en este contexto.
            For Each prpBucle In .Properties
               On Error Resume Next
               If prpBucle <> "" Then Debug.Print _
                  "      " & prpBucle.Name & _
                  " = " & prpBucle
               On Error GoTo 0
            Next prpBucle
         End With
      Next rstBucle
      rstTable.Close
      rstDynaset.Close
      rstSnapshot.Close
      rstForwardOnly.Close
.Close
   End With
End Sub
```

- *BoundColumn*: **ISBN.**
- *ListField*: **nombreRevista .**

EL CONTROL DBGRID

Todos los formularios tienen un botón con el título *Grid*, el cual abre el formulario **frmDataGrid.** Este contiene *DBGrid* para visualizar campos de una tabla de forma similar a las tablas utilizadas en *Access*. Para conseguir que la rejilla presente los datos únicamente hay que conectar con un control de datos mediante la propiedad *DataSource*.

Correo del lector

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

Pregunta

Ante todo agradeceros el gran nivel que soporta vuestra publicación. A partir de ella me lancé al uso de Linux, y desde ese momento sólo uso este S.O., excepto Windows para cosas que sólo se pueden hacer en Windows.

Tengo unas preguntas que haceros:

1) ¿Se pueden capturar los mensajes printk a un fichero?

2) Estoy intentando configurar una conexión PPP en EarthLink, pero no me puedo conectar porque parece que no existe DNS alguna. Asigno una IP y conecta, pero NetScape y ncftp no son capaces de encontrar ninguna otra dirección (resolve names error). ¿Cómo puedo entrar el nombre del servidor EarthLink (o su IP) y configurarlo adecuadamente?

3) ¿Cómo imprimo las páginas de ayuda obtenidas con el comando man?

Muchas gracias por el interés demostrado en mis preguntas.

Respuesta

Ante todo agradecerte la confianza que depositas en la revista.

Con respecto a tus preguntas, quizás sería más conveniente que nos indicaseis la versión que tenéis instalada en vuestro ordenador de LINUX. Eso nos ayudaría a poder localizar con más exactitud vuestras dudas. De todos modos, intentaré solventar tus problemas.

1) Quizás deberíamos hacer uso de algunas de las ventajas que Linux nos ofrece. Por ejemplo, podrías usar las facilidades que syslog() nos ofrece. Desde el kernel, podríamos usar sys_syslog(), y desde allí, nos resultará fácil configurar el demonio (daemon) syslogd() para que redirija las salidas a un fichero diferente del standard /var/log/ messages.

2) La verdad es que no me das demasiados datos, pero espero poderte ayudar:

Tenemos dos modos de resolver tu problema.

a) Si sólo tienes una ISP y la DNS es siempre la misma, puedes probar a poner lo siguiente en el fichero /etc/resolv.conf:

```
Domain earthlink.net
Nameserver 10.10.10.10
```

Por supuesto, deberás reemplazar 10.10.10.10 por tu dirección IP.

b) Si usas X Window, puedes añadir el adaptador a través de Administration > Network Config menu

De este modo obtienes un GUI que puede ser usado para configurar el adaptador y usar GW y NS en la otra parte del PPP.

3) Muy sencillo: man -t manpage | lpr
Esto en ps version.

Pregunta

Hola. Soy un subscriptor de vuestra revista prácticamente desde los comienzos de ésta, y después de programar durante bastantes años en Visual Basic he creído lo más interesante y conveniente pasarme a Delphi. Así que tengo que haceros unas cuestiones que confío en que puedean resolver mis dudas. Mi primera pregunta sería la siguiente: Actualmente uso Delphi 3 Professional, y ante el

lanzamiento de Delphi 4 creo que migraré, ¿Qué me recomendáis, actualizar a Delphi 4 Professional o pasarme a la Cliente/Servidor? Estoy usando Interbase Local para mis aplicaciones, y tengo bastantes problemas a la hora de poner los campos a Null. ¿Me podríais echar una mano?

Respuesta

Vamos a ir por partes. El uso de las versiones de Delphi profesionales Cliente/Servidor depende mucho de las necesidades que tengas como programador. Yo particularmente prefiero normalmente casi siempre las versiones Cliente/Servidor, porque ofrecen multitud de ventajas sobre la profesional.

En principio Delphi 4 soporta objetos Corba, todo lo relativo a MIDAS y objetos COM, y una paleta de componentes mucho más rica. En contra, el precio, que es siempre muy superior. Desde luego, crear aplicaciones distribuidas y Cliente/Servidor es más sencillo con la versión Cliente/Servidor que con la profesional, pero si tu presupuesto es reducido, siempre podrás usar la versión profesional y trabajar 'simulando' la versión Cliente/ Servidor (una locura, por cierto).

La segunda pregunta tiene fácil solución. Si lo que estás haciendo es 'limpiando' a NULL los campos de una tabla, te basta con llamar al método Clear si estás en modo inserción o modificación. Si estás usando TQuery's y no usas parámetros, te basta con pasar como valor NULL (trabajas con Interbase), pero si usas parámetros para los TQuery's la cosa se complica un poco. Sírvate este ejemplo como modelo:

```
With QInsertar do  {TQUery definido en un
      Form}
 Begin
Close;
      Sql.clear;
      Sql.add('INSERT INTO TABLA
      (CAMPO1,CAMPO2) VALUES (:UNO,
      :DOS)');
      Prepare; {recordemos, usaremos un
      ExecSql}
ParamByName('DOS').DataType := ftstring; {es
      un TParam}
      ParamByName('UNO').AsString :=
      'FRANCISCO ROSADO';
      ParamByName('DOS').Clear;
      ExecSql;
 End;
```

De este modo, y siempre definiendo el tipo de parámetro , podemos pasar un valor NULL a un TQuery.

Pregunta

Soy un habitual lector de vuestra revista, y después de mucho preguntar a amigos y compañeros me he decidido a enviaros una consulta. Mi pregunta es simple: ¿Cómo puedo lanzar desde Java una URL determinada, es decir, abrir otro navegador apuntando a una URL determinada? Lo he intentado montones de veces y me resulta imposible. Gracias.

Respuesta

Bueno, tu respuesta tiene una solución , como casi todo, que para Java se llama JavaScript. La función, aunque mejorable, puede ser como la siguiente:

```
function newWindow {
      newWindow = window.open("El URL",
      "nombre de la nueva ventana",
      "toolbar=yes,resizable=yes,más conside-
      raciones, Francisco Rosado para SOLO
      PROGRAMADORES");
      }
```

Después te basta con llamar a esta función desde tu applet usando la función Netscape JSObject del paquete del mismo nombre.

Espero haberte sido de ayuda.

Pregunta

He leído un artículo en la revista Sólo Programadores sobre creación de componentes en C++ Builder el cual ha sido de mi interés, pues he usado mucho el C++ de Borland.

Tengo unas preguntas para ustedes, si no les importa:

Tengo una aplicación de hace ya algún tiempo hecha en Borland C++ 4.0 basada en OWL, para Windows 3.1. Me gustaría portarla y mejorarla, pues ya ha llovido desde que salió el 4.0... ¿Cómo de sencillo me podría resultar portarla al C++ Builder para continuarla en este sistema para Windows 95/98/NT ?

He visto por ahí que también hay Borland C++ 5.x. ¿ Qué diferencia existe con el C++ Builder ?

Respuesta

Ante todo quiero agradecerte que leas con tanto interés los artículos. Esto llena de satisfacción a todo escritor. Si además es profesional del sector informático, aún nos llena más.

Con respecto a tus preguntas, la migración C++ -> C-Builder es similar a la que puede existir Turbo Pascal -> Delphi. Puedes coger tu aplicación de C++ y compilarla perfectamente desde el IDE de C-Builder, pero, realmente no habrás hecho más que compilarla en C-Builder. La aplicación sería la misma.

Como recomendación, y ante este paso de migración que quieres

dar, parece lógico que te plantees un estudio de la aplicación y un nuevo diseño. Como sabes, las bases de datos avanzan una barbaridad, los componentes, los objetos, ..., con lo que parece lógico que evites todo intento de una aplicación igual.

Lógicamente, todos los .CPP que tengas con funciones, objetos, librerías, ... los puedes usar sin tener que escribir de nuevo el código. Eso sí que te vale. Pero las pantallas, por ejemplo, nada de nada.

Con respecto a Borland C++ 5.0, es cierto que existe en el mercado. De echo, si compras C-Builder 3.0 C/S te lo regalan. Todo esto después de que revistas de las más prestigiosas del sector informático juraran y perjuraran la muerte de C++. Ya conoces C++. La versión 5.0 añade nuevas funcionalidades, mayor velocidad y un entorno más amigable aún si cabe. Las diferencias con C-Builder son muchísimas. Piensa que son filosofías de programación totalmente diferentes. Sirve de ejemplo el uso de componentes y forms, y su 'no uso'.

Pregunta

Ante todo me gustaría poder agradeceros el esfuerzo que mes a mes ponéis en la revista, para que nos llegue a los lectores lo mejor de lo mejor del contenido informático . Me parece sin lugar a dudas un producto de gran calidad y de envidiable presentación. Además quiero aprovechar la oportunidad que brindáis a los lectores ,como yo, en esta sección, nuestra sección, para haceros llegar una pregunta sobre Delphi. Actualmente yo estoy utilizando Delphi 4.0 y QR 3.0, y desearía saber si existe alguna manera de poder imprimir un Dataset directamente, es decir, si QR dispone de alguna plantilla ya predefinida con un listado. Como ya os podréis imaginar, para mi es una lata tener que hacer para cada mantenimiento un listado diferente.

Respuesta

Gracias por tus cumplidos. Con respecto a tus preguntas, debo decirte que tienen solución a partir de la versión 3 de Quick Report. En concreto la unidad QrExtra nos permite lanzar un listado predefinido.

Usaremos la función QRCreateList, a la cual se le pasan una serie de parámetros:

1) una variable que definamos de tipo TCustomQuickRep. Si le pasamos a la función nil, se crea el objeto. Si le pasamos un objeto, se intenta aprovechar.

2) El padre del objeto listado.

3) Un conjunto de datos sobre los que se basará el informe.

4) El título del listado.

5) Un TString con los campos que queremos que aparezcan en el listado. Si no indicamos nada, suponemos que se usaran todos los campos.

Un ejemplo completo podría constar de lo siguiente:

```
Procedure LanzaListado;
Var
  Listado :TCustomQuickRep;
Campos : TStrings;
Begin
{cargado de los campos en el TString}
...
{El lanzamiento del listado}
Listado := nil;
  Try
    QrCreateList(Listado, self, Tabla1, 'Solo
      programadores',Campos)
  Finally
   .....
  End;
End;
```

De este modo tan sencillo obtenemos un listado genérico que podemos usar en todos nuestros mantenimientos de catálogos.

Pregunta

He empezado a programar en Java y como principiante que soy, no consigo comprender el uso de Sockets desde la función de librería java.net. Estaría encantado si pudiéseis responderme, ya que me encuentro como habitualmente se suele decir "sólo ante el peligro". Muchas gracias por vuestra ayuda.

Respuesta

Los sockets deben funcionar ya estén los procesos en la misma máquina o no. Se intercambian datos a través de cualquier tipo de red, local o remota.

Bajo esto, un proceso servidor declara aceptar las conexiones de uno o varios clientes, y cuando el cliente aparece, se crea el socket para comunicarse entre ambos. Java ofrece las clases ServerSocket y Socket.

Los atributos de la clase servidor son un objeto del tipo ServerSocket, que sirve para esperar a las conexiones, y un objeto de tipo Socket que sirve para gestionar una conexión a la vez.

Nos bastaría con asignar a la función ServerSocket el número de un puerto, del modo

```
Servidor = new ServerSocket(1998);
Esperar al próximo cliente:
Cliente = Servidor.accept();
```

Después usar los ficheros de entrada y salida asociados al objeto de tipo Socket.

lanzamiento de Delphi 4 creo que migraré, ¿Qué me recomendáis, actualizar a Delphi 4 Professional o pasarme a la Cliente/Servidor? Estoy usando Interbase Local para mis aplicaciones, y tengo bastantes problemas a la hora de poner los campos a Null. ¿Me podríais echar una mano?

Respuesta

Vamos a ir por partes. El uso de las versiones de Delphi profesionales Cliente/Servidor depende mucho de las necesidades que tengas como programador. Yo particularmente prefiero normalmente casi siempre las versiones Cliente/Servidor, porque ofrecen multitud de ventajas sobre la profesional.

En principio Delphi 4 soporta objetos Corba, todo lo relativo a MIDAS y objetos COM, y una paleta de componentes mucho más rica. En contra, el precio, que es siempre muy superior. Desde luego, crear aplicaciones distribuidas y Cliente/Servidor es más sencillo con la versión Cliente/Servidor que con la profesional, pero si tu presupuesto es reducido, siempre podrás usar la versión profesional y trabajar 'simulando' la versión Cliente/ Servidor (una locura, por cierto).

La segunda pregunta tiene fácil solución. Si lo que estás haciendo es 'limpiando' a NULL los campos de una tabla, te basta con llamar al método Clear si estás en modo inserción o modificación. Si estás usando TQuery's y no usas parámetros, te basta con pasar como valor NULL (trabajas con Interbase), pero si usas parámetros para los TQuery's la cosa se complica un poco. Sírvate este ejemplo como modelo:

```
With QInsertar do  {TQUery definido en un
     Form}
 Begin
 Close;
      Sql.clear;
      Sql.add('INSERT INTO TABLA
      (CAMPO1,CAMPO2) VALUES (:UNO,
      :DOS)');
      Prepare; {recordemos, usaremos un
      ExecSql}
ParamByName('DOS').DataType := ftstring; {es
      un TParam}
      ParamByName('UNO').AsString :=
      'FRANCISCO ROSADO';
      ParamByName('DOS').Clear;
      ExecSql;
 End;
```

De este modo, y siempre definiendo el tipo de parámetro , podemos pasar un valor NULL a un TQuery.

Pregunta

Soy un habitual lector de vuestra revista, y después de mucho preguntar a amigos y compañeros me he decidido a enviaros una consulta. Mi pregunta es simple: ¿Cómo puedo lanzar desde Java una URL determinada, es decir, abrir otro navegador apuntando a una URL determinada? Lo he intentado montones de veces y me resulta imposible. Gracias.

Respuesta

Bueno, tu respuesta tiene una solución , como casi todo, que para Java se llama JavaScript. La función, aunque mejorable, puede ser como la siguiente:

```
function newWindow {
      newWindow = window.open("El URL",
      "nombre de la nueva ventana",
      "toolbar=yes,resizable=yes,más conside-
      raciones, Francisco Rosado para SOLO
      PROGRAMADORES");
      }
```

Después te basta con llamar a esta función desde tu applet usando la función Netscape JSObject del paquete del mismo nombre.

Espero haberte sido de ayuda.

Pregunta

He leído un artículo en la revista Sólo Programadores sobre creación de componentes en C++ Builder el cual ha sido de mi interés, pues he usado mucho el C++ de Borland.

Tengo unas preguntas para ustedes, si no les importa:

Tengo una aplicación de hace ya algún tiempo hecha en Borland C++ 4.0 basada en OWL, para Windows 3.1. Me gustaría portarla y mejorarla, pues ya ha llovido desde que salió el 4.0... ¿Cómo de sencillo me podría resultar portarla al C++ Builder para continuarla en este sistema para Windows 95/98/NT ?

He visto por ahí que también hay Borland C++ 5.x. ¿ Qué diferencia existe con el C++ Builder ?

Respuesta

Ante todo quiero agradecerte que leas con tanto interés los artículos. Esto llena de satisfacción a todo escritor. Si además es profesional del sector informático, aún nos llena más.

Con respecto a tus preguntas, la migración C++ -> C-Builder es similar a la que puede existir Turbo Pascal -> Delphi. Puedes coger tu aplicación de C++ y compilarla perfectamente desde el IDE de C-Builder, pero, realmente no habrás hecho más que compilarla en C-Builder. La aplicación sería la misma.

Como recomendación, y ante este paso de migración que quieres

dar, parece lógico que te plantees un estudio de la aplicación y un nuevo diseño. Como sabes, las bases de datos avanzan una barbaridad, los componentes, los objetos, ..., con lo que parece lógico que evites todo intento de una aplicación igual.

Lógicamente, todos los .CPP que tengas con funciones, objetos, librerías, ... los puedes usar sin tener que escribir de nuevo el código. Eso sí que te vale. Pero las pantallas, por ejemplo, nada de nada.

Con respecto a Borland C++ 5.0, es cierto que existe en el mercado. De echo, si compras C-Builder 3.0 C/S te lo regalan. Todo esto después de que revistas de las más prestigiosas del sector informático juraran y perjuraran la muerte de C++. Ya conoces C++. La versión 5.0 añade nuevas funcionalidades, mayor velocidad y un entorno más amigable aún si cabe. Las diferencias con C-Builder son muchísimas. Piensa que son filosofías de programación totalmente diferentes. Sirve de ejemplo el uso de componentes y forms, y su 'no uso'.

Pregunta

Ante todo me gustaría poder agradeceros el esfuerzo que mes a mes ponéis en la revista, para que nos llegue a los lectores lo mejor de lo mejor del contenido informático . Me parece sin lugar a dudas un producto de gran calidad y de envidiable presentación. Además quiero aprovechar la oportunidad que brindáis a los lectores ,como yo, en esta sección, nuestra sección, para haceros llegar una pregunta sobre Delphi. Actualmente yo estoy utilizando Delphi 4.0 y QR 3.0, y desearía saber si existe alguna manera de poder imprimir un Dataset directamente, es decir, si QR dispone de alguna plantilla ya predefinida con un listado. Como ya os podréis imaginar, para mi es una lata tener que hacer para cada mantenimiento un listado diferente.

Respuesta

Gracias por tus cumplidos. Con respecto a tus preguntas, debo decirte que tienen solución a partir de la versión 3 de Quick Report. En concreto la unidad QrExtra nos permite lanzar un listado predefinido.

Usaremos la función QRCreateList, a la cual se le pasan una serie de parámetros:

1) una variable que definamos de tipo TCustomQuickRep. Si le pasamos a la función nil, se crea el objeto. Si le pasamos un objeto, se intenta aprovechar.

2) El padre del objeto listado.

3) Un conjunto de datos sobre los que se basará el informe.

4) El título del listado.

5) Un TString con los campos que queremos que aparezcan en el listado. Si no indicamos nada, suponemos que se usaran todos los campos.

Un ejemplo completo podría constar de lo siguiente:

```
Procedure LanzaListado;
Var
  Listado :TCustomQuickRep;
Campos : TStrings;
Begin
{cargado de los campos en el TString}
...
{El lanzamiento del listado}
Listado := nil;
  Try
    QrCreateList(Listado, self, Tabla1, 'Solo
      programadores',Campos)
  Finally
   .....
  End;
End;
```

De este modo tan sencillo obtenemos un listado genérico que podemos usar en todos nuestros mantenimientos de catálogos.

Pregunta

He empezado a programar en Java y como principiante que soy, no consigo comprender el uso de Sockets desde la función de librería java.net. Estaría encantado si pudiéseis responderme, ya que me encuentro como habitualmente se suele decir "sólo ante el peligro". Muchas gracias por vuestra ayuda.

Respuesta

Los sockets deben funcionar ya estén los procesos en la misma máquina o no. Se intercambian datos a través de cualquier tipo de red, local o remota.

Bajo esto, un proceso servidor declara aceptar las conexiones de uno o varios clientes, y cuando el cliente aparece, se crea el socket para comunicarse entre ambos. Java ofrece las clases ServerSocket y Socket.

Los atributos de la clase servidor son un objeto del tipo ServerSocket, que sirve para esperar a las conexiones, y un objeto de tipo Socket que sirve para gestionar una conexión a la vez.

Nos bastaría con asignar a la función ServerSocket el número de un puerto, del modo

```
Servidor = new ServerSocket(1998);
Esperar al próximo cliente:
Cliente = Servidor.accept();
```

Después usar los ficheros de entrada y salida asociados al objeto de tipo Socket.